Relational Extensions to Feature Logic : Applications to Constraint Based Grammars

PhD Thesis

Suresh K Manandhar

Department of Artificial Intelligence
Faculty of Science and Engineering
University of Edinburgh

1993

Declaration

I hereby declare that I composed this thesis entirely myself and that it describes my own research.

Abstract

This thesis investigates the logical and computational foundations of unification-based or more appropriately constraint based grammars. The thesis explores extensions to feature logics (which provide the basic knowledge representation services to constraint based grammars) with multi-valued or relational features. These extensions are useful for knowledge representation tasks that cannot be expressed within current feature logics.

The approach bridges the gap between concept languages (such as KL-ONE), which are the mainstay of knowledge representation languages in AI, and feature logics. Various constraints on relational attributes are considered such as existential membership, universal membership, set descriptions, transitive relations and linear precedence constraints.

The specific contributions of this thesis can be summarised as follows:

1. Development of an integrated feature/concept logic
2. Development of a constraint logic for so called partial set descriptions
3. Development of a constraint logic for expressing linear precedence constraints
4. The design of a constraint language CL-ONE that incorporates the central ideas provided by the above study
5. A study of the application of CL-ONE for constraint based grammars

The thesis takes into account current insights in the areas of constraint logic programming, object-oriented languages, computational linguistics and knowledge representation.

Acknowledgements

Firstly I would like to thank Chris Mellish. This work would not have been possible without him laboriously going through my draft Chapters and always asking for new ones. His guidance and encouragement has added greatly to this work. Alan Smaill has helped me greatly by checking my proofs, finding out the errors and suggesting corrections. I am greatly indebted to Pete Whitelock who initially gave me lot of papers to read and suggested this area of research. I would also like to thank my two examiners, Ewan Klein and Hans Uszkoreit, for suggestions and comments.

I owe a great deal to friends and colleagues within the Edinburgh academic community. Particular thanks go to - Alan Black, Matt Crocker, Ian Lewin and Mike Reape for many helpful discussions. I also cannot forget my friends who has made my time in Edinburgh enjoyable and memorable: John Beaven, Andy Bowles, Flávio Corrêa da Silva, Alvaro Fernandes, Regina Fernandes, Carla Pedro Gomes, Ian Frank, Nelson Ludlow, Andrei Mikheev, Dave Moffat, Keiichi Nakata, Brian Ross, Rob Scott, Wamberto Vasconcelos and others who have passed through E17.

I would also like to thank Elisabet Engdahl, Chris Brew, Marc Moens, Drew Moshier for helpful comments and discussions.

I am also indebted to various funding bodies who have made my studies possible : the Overseas Research Scheme Award, an Edinburgh University Studentship, and finally a research assistantship in the Human Communication Research Centre under the LRE project 61061 and the Department of Artificial Intelligence which allowed me to travel to various workshops and Summer Schools.

I would like to thank both the computing support staff at the Department of Artificial Intelligence, the AIAI and the Human Communication Research Centre for maintaining an excellent computing environment. I would like to thank Richard Cayley and Lex Holt for helping me with LaTex macros.

Finally I am greatly indebted to my parents and my brother who have given me tremendous moral support and encouragement. This work would not have been possible without Am whose has provided me with love, care, understanding and support.

Contents

Chapter 1

Introduction

This thesis presents a systematic study of extensions to feature logics - for their incorporation into a knowledge representation formalism and investigates the applicability of such a formalism for the description of constraint based grammars.

1.1 Thesis Aims

The principal aim of the thesis is to extend the expressiveness of feature logic with constraints involving relational attributes and explore the applications of such a extended logic both for an integrated knowledge representation formalism and for constraint based grammar formalisms. This work is intended as a major step in the goal of designing a knowledge representation formalism that is suitable for both general A.I. needs and for computational linguistic applications.

1.1.1 Thesis Contributions

The specific contributions of this thesis can be summarised as follows:

1. The thesis develops an integrated feature/concept logic dubbed $\mathcal{ALV}$ that effectively integrates feature logic and the terminological component of concept languages such as KL-ONE. This is an important contribution since it is known that a straightforward integration of feature logic and terminological logic leads to *undecidability*. This due to the fact that while the logical construct known as

path equations is an important construct for unification grammars and does not cause *undecidability*, its relational counterpart, known as *role-value maps*, causes *undecidability* in KL-ONE [Schmidt-Schauß 89]. This means that the straightforward route for integrating a variable-free feature term language [Smolka 88] with a variable-free concept language such as $\mathcal{ALC}$ [Schmidt-Schauß & Smolka 91] would cause undecidability.

This thesis shows that eliminating path equations but introducing *unquantified variables* in an integrated propositionally complete concept language that provides both features and relations does not cause *undecidability.*

2. The thesis investigates the logical and computational foundation of the so called set descriptions which are being used in computational linguistics. Although several formalisations of set descriptions are known [Pollard & Moshier 90], [Rounds 88] their integration into feature logic has not yet been accomplished nor has there been an rigorous study of consistency checking techniques for an extended feature logic that provides both features and set descriptions.

   We provide a detailed picture of the semantics of set descriptions and rigorously establish their connection with concept languages containing the construct known as *number restrictions* [Hollunder & Nutt 90]. Our formalisation not only makes precise what set descriptions mean but also provides a semantics for negative set descriptions. We extend consistency checking techniques to formulae involving set descriptions.

3. We explore the addition of transitive relations to a positive fragment of our extended logic (not involving *disjunctions* or *negations*) with the aim of providing an interpretation of *linear precedence* constraints which are crucially required in the specification of word order variation in natural languages. Our approach provides a starting point for a general framework for treating transitive relations and precedence constraints.

4. We provide a detailed design study for a knowledge representation language dubbed CL-ONE based mainly on the theoretical study mentioned earlier. This study shows the shape of a future knowledge representation framework that is expressly

designed to accommodate enhanced knowledge representation needs of current constraint based grammars which are unavailable in current knowledge representation languages. Substantial fragments of CL-ONE have been implemented mainly with the aim of verifying feasibility.

5. We investigate systematically the applicability of the constraint language CL-ONE for linguistic description. Our study describes methods for encoding thematic roles and proto-roles in lexical entries, for the specification of thematic role based argument selection principles and proto-role based subject selection principles. The final application is an HPSG based specification of DRT.

1.2 Background and Related Work

In this section we provide an informal survey of *feature structures, feature logic, typed feature logic and formalisms, constraint based grammars* and *concept languages.* These provide the background material needed to understand the rest of the thesis. However, neither completeness of the survey nor mathematical accuracy is intended. Rather the aim has been to provide an intuitive description of the subject material. The interested reader should refer to the references that are cited for a more accurate description.

1.2.1 Feature Structures

Feature structures are the basic datastructure employed both in conventional unification grammars [Shieber 84] [Gazdar *et al* 85] and in constraint based grammars such as HPSG [Pollard & Sag 87]. Intuitively speaking, feature structures are just a labelled representation of tuples. For instance, a feature structure representation of the tuple:

$$(male, 28)$$

could be the feature structure (represented in *Prolog* notation):

$$[sex : male, age : 28]$$

where *sex* and *age* are called *feature labels* and *male* and *age* are the *values* of the corresponding labels.

Linguists have developed a convenient graphical notation known as the *attribute value matrix* (AVM) box for describing feature structures (without committing to a specific programming language notation). An AVM box for describing the above example is given below.

(1) **AVM box notation**

$$\begin{bmatrix} \text{SEX} & \textit{male} \\ \text{AGE} & 28 \end{bmatrix}$$

Feature Structure Unification

Central to feature structures is the notion of *feature structure unification.* Feature structure unification is the analog of term unification of first-order terms.

Just like the unification of the first-order term:

$$person(male, Y)$$

with the term:

$$person(X, 28)$$

results in the term:

$$person(male, 28)$$

with the variable bindings:

$$X = male, Y = 28$$

Similarly, the unification of the feature structure:

$$\begin{bmatrix} \text{PERSON} & \textit{male} \end{bmatrix}$$

with the feature structure:

$$\begin{bmatrix} \text{AGE} & 28 \end{bmatrix}$$

results in the feature structure

(2) $$\begin{bmatrix} \text{PERSON} & \textit{male} \\ \text{AGE} & 28 \end{bmatrix}$$

One difference between feature structures and first-order terms is the lack of a fixed arity in the former. In this sense, feature structures can be thought of as terms with infinite arity.

A central notion to feature structures is that feature labels are interpreted as *functional.* Thus for instance the unification of the feature structures:

$$\left[\text{PERSON}\left[\begin{array}{ll}\text{NAME} & \textit{john}\\ \text{AGE} & 28\end{array}\right]\right] \text{ with } \left[\text{PERSON}\left[\begin{array}{ll}\text{NAME} & X\\ \text{SALARY} & 12\end{array}\right]\right]$$

results in the feature structure:

$$\left[\text{PERSON}\left[\begin{array}{ll}\text{NAME} & \textit{john}\\ \text{AGE} & 28\\ \text{SALARY} & 12\end{array}\right]\right] \text{ where } X = \textit{john}$$

while the unification of feature structures

$$\left[\text{PERSON}\left[\text{NAME } \textit{john}\right]\right] \text{ with } \left[\text{PERSON}\left[\text{NAME } \textit{mary}\right]\right]$$

will *fail.*

Finally, *atoms* (*e.g. john, mary, 12 etc.*) are considered to be those kind of structures on which *no* feature is defined. Thus unifying $\left[\text{AGE } 28\right]$ with the atom *john* results in *failure.*

1.2.2 Feature logic

Feature logic is a simple logic for which feature structures are the models. More simply, feature structures can be thought of as the objects (though not necessarily the only ones) that formulae in feature logic describe.

Feature logic has been formalised by several researchers, notably [Kasper & Rounds 86], [Moshier & Rounds 87], [Johnson 88] and in a general setting by [Smolka 88]. Within the feature term language (which can be thought of as an instance of feature logic) of [Smolka 88], for instance, a formula for describing *a male aged 28* would be by the

feature term:

$$sex : male \sqcap age : 28$$

for which the feature structure given in (2) is an appropriate model.

The symbol $\sqcap$ is to be understood as *conjunction* or *intersection.* Feature terms come equipped with all the propositional connectives. In other words, the connectives $\sqcup$ understood as *disjunction* or *union* and $\neg$ understood as *negation* or *complement* are also available.

Feature terms are interpreted as denoting the collection of all the feature structures that satisfy the term. Thus the feature term $sex : male$ can be used describe the following feature structures among a possible infinite number of others.

(3)

formulae		Interpretation
$sex : male$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{SEX } male \end{bmatrix}$
$sex : male$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{SEX } male \\ \text{AGE } 28 \end{bmatrix}$
$sex : male$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{SEX } male \\ \text{AGE } 29 \end{bmatrix}$

Thus similarly, the feature term $age : 28$ would among others model the following feature structures.

(4)

formulae		Interpretation
$age : 28$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{AGE } 28 \end{bmatrix}$
$age : 28$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{SEX } male \\ \text{AGE } 28 \end{bmatrix}$
$age : 28$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{SEX } female \\ \text{AGE } 28 \end{bmatrix}$
$age : 28$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{SEX } anything \\ \text{AGE } 28 \end{bmatrix}$

The denotation of a feature term (for a given interpretation $\mathcal{I}$) such as $sex : male$ denoted by $[\![sex : male]\!]^{\mathcal{I}}$ is then given as the collection of all the models (*i.e.* feature structures) that satisfy $sex : male$. In other words:

(5) $[\![sex : male]\!]^{\mathcal{I}} =$ all models of $sex : male$

Atoms are interpreted as denoting singleton sets. Thus the denotation of the atom *male* would be given by $[\![male]\!]^{\mathcal{I}} = \{male^{\mathcal{I}}\}$.

Yet another type of construct that most variants of feature logic come equipped with is known as *path equations*. A path equation is a term of the form:

(6) $P = Q$

where P and Q are lists of features *i.e.* they have the form $f_1 \ldots . f_n$ where each of the f_i's are feature symbols.

A path equation $P = Q$ is interpreted as the (collection of) feature structures whose P-value is identical (or co-referential) to its Q-value. In other words:

(7) $[\![P = Q]\!]^{\mathcal{I}} =$ all models of $\mathcal{I}$ such that their P and Q values are co-referential

Thus the possible interpretations of a term $f = g$ can be given as follows:

(8)

formulae		Interpretation
$f = g$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{F } 20 \\ \text{G } 20 \end{bmatrix}$
$f = g$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{F } \boxed{1} \begin{bmatrix} \text{SEX } male \\ \text{AGE } 29 \end{bmatrix} \\ \text{G } \boxed{1} \end{bmatrix}$

The boxed number $\boxed{1}$ above is a commonly used AVM notation which indicates that the values of the feature labels F and G are identical.

The interpretation of conjunctions ($\sqcap$), disjunctions ($\sqcup$) and negations ($\neg$) can then be given as follows:

1. $[\![sex : male \sqcap age : 28]\!]^{\mathcal{I}} = [\![sex : male]\!]^{\mathcal{I}} \cap [\![age : 28]\!]^{\mathcal{I}}$
2. $[\![sex : male \sqcup age : 28]\!]^{\mathcal{I}} = [\![sex : male]\!]^{\mathcal{I}} \cup [\![age : 28]\!]^{\mathcal{I}}$
3. $[\![\neg sex : male]\!]^{\mathcal{I}} =$ all models of $\mathcal{I}$ - $[\![sex : male]\!]^{\mathcal{I}}$

Thus intersections (conjunctions) are interpreted as set intersections, unions (disjunctions) are interpreted as set unions and complements (negations) are interpreted as set complements.

Thus, from the above definitions, it follows that the feature structure:

(9) $\begin{bmatrix} \text{SEX } male \\ \text{AGE } 28 \end{bmatrix}$

is a model for the feature term:

(10) $sex : male \sqcap age : 28$

In fact the feature structure given in (9) is a *canonical model* (or *minimal model*) for the above feature term.

Formalised in this fashion, feature logic is *propositionally complete* which means that it respects all the theorems of *propositional logic.*

Within feature logic the notion of unification is replaced with the more general notion of *consistency* of a given formula.

The following (informal) definitions characterise the notions of *consistency, inconsistency* and their relationship to feature structure *unification.*

- Formula S is **consistent** if we can find an interpretation $\mathcal{I}$ such that $[\![S]\!]^{\mathcal{I}}$ is non-empty.
- Formula S is **inconsistent** if $[\![S]\!]^{\mathcal{I}}$ is empty in every interpretation. *e.g.* $sex : 28 \sqcap sex : 29$ is *inconsistent.*
- *Unification* of S with T has the property of **succeeding** if $S \sqcap T$ is *consistent.*

- *Unification* of S with T **fails** if $S \sqcap T$ is *inconsistent.*

This discussion completes an informal introduction to feature logic.

Limitations of Feature Logic

One of the central limitations of feature logic that we want to address in this thesis is the fact that feature labels are interpreted as *functional* (or *single-valued*) within feature logic. Thus it is impossible to state within feature logic a fact such as *John has Kim and Alan as children.* On the other hand, within frame based languages of the KL-ONE family attribute labels are interpreted relationally (*see* section 1.2.6). Furthermore, in section 1.3.3 we show that current constraint based grammars employ descriptive machinery whose formalisation requires the use of multi-valued or relational attributes.

1.2.3 Constraint solvers and consistency checking

Consistency checking is a basic computational service that any given feature logic based system should provide. Normally speaking the task of consistency checking is delegated to a *constraint solver* whose primary job is to test the consistency of a given formula or set of constraints. The job of the constraint solver is to ensure that the constraint system it is maintaining is consistent as new constraints get *incrementally* added to the constraint system. Constraint solvers are usually designed to perform only a minimal amount of (re-)computation as new constraints gets added. This is known as *incremental constraint solving* and is an important and distinguishing characteristic of constraint solvers that distinguishes them from ordinary *theorem provers.*

A *constraint system* is normally a set of constraints interpreted as a *conjunction* of simple constraints or formulae. One important property of constraint systems is *monotonicity* in that the addition of new constraints can only render a *consistent* constraint system *inconsistent* but *not* vice versa. Thus as soon as an inconsistency is detected the constraint solver can simply discard all the constraints.

Constraint solvers are the heart of computational systems for implementing constraint

based grammars. This makes the study of constraint solving techniques an important requirement for constructing efficient knowledge representation languages for NLP. The approach taken in this thesis is to extend known constraint solving techniques for concept languages [Schmidt-Schauß & Smolka 91] [Donini *et al* 91] and that for feature logic [Smolka 89] to the class of extended feature/concept logics studied in this thesis.

1.2.4 Typed feature logic based formalisms

For applications which need to store logical definitions and make repeated use of these definitions, the logical machinery available in feature logic alone in insufficient. For instance, we may need to store the information that every *man* is a *human being of male sex.* Thus we would want a definitional mechanism such as illustrated by the following examples:

$$Man = Human \sqcap sex : male$$

$$Woman = Human \sqcap sex : female$$

The symbols Man, $Woman$ and $Human$ are called *type* (or *sort*) symbols[1].

Note that according to the above definitions the formula:

$$Man \sqcap Woman$$

is *inconsistent.*

The resultant *hybrid* logic that integrates a feature logic with a type definition mechanism is usually referred to as *typed feature logic.* The notion of a type definition mechanism extends not just to feature logics but to other logics such as concept languages and is closely related to Horn clause definitions[Höhfeld & Smolka 88].

The computational systems that implement typed feature logics are known as *typed feature formalisms.* The formalisms TFS [Zajac 92], STUF [Dörre & Seiffert 91], LO-

[1]In this thesis we shall refer to *types* as meaning symbols that have a definition and *sorts* loosely meaning symbols which either may or may not have a definition. We shall be explicit when *sorts* mean symbols that do not have a definition.

GIN [Aït-Kaci & Nasr 86], LIFE [Aït-Kaci & Podelski 91], CUF [Eisele & Dörre 90] and ALE [Carpenter 93] are prime examples of existing typed feature formalisms.

Constraint based grammatical frameworks such as HPSG are based around the idea of being able to specify grammars within a (possibly extended) typed feature logic based formalism.

A set of type definitions is referred to as a *terminology*[2]. A *terminology* is *cyclic* if it contains definitions which are (ultimately) defined in terms of themselves. For instance, the following terminology is *cyclic.*

$$Man = Human \sqcap \neg Woman$$

$$Woman = Human \sqcap \neg Man$$

Cyclic terminologies present difficulties both in their interpretation and their computational feasibility. There are several known characterisations of cyclic terminologies namely the *least-fixed point, greatest-fixed point* [Baader 90] and *descriptive* semantics [Nebel 91]. Thus for instance, the definition:

$$btree = left : btree \sqcap right : btree$$

under the least fixed-point interpretation would only permit infinite models of the form given in figure 1.1. On the other hand, under the greatest fixed-point interpretation, only cyclic models of the form given in 1.2 would be admitted. And finally a descriptive semantics would permit any of the above as models being the most permissive among the three.

The notion of typing extends to feature labels too. For instance, one may want to enforce the condition that the feature label *sex* is a function from the sort *Human* to the sort *Sex* (distinct from the feature label *sex*), hence applying the feature label *sex* to something other than *Human* would lead to an inconsistency. Such a specification would be of the form:

[2]We follow the notation employed in the concept language literature here (see for instance [Nebel & Smolka 91], [Nebel 90]).

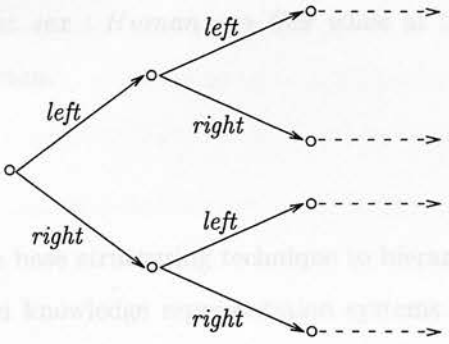


Figure 1.1: Infinite Model

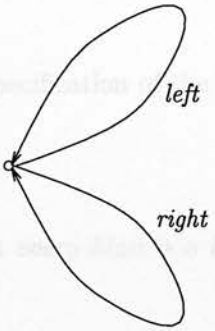


Figure 1.2: Cyclic Model

$$sex : Human \longrightarrow Sex$$

Thus (under some assumptions about the way relation typing specifications are interpreted) the descriptions:

$$sex : \top \quad \text{and} \quad Human \sqcap sex : Sex$$

are *equivalent i.e.* they have the same denotations in every interpretation that respect the *feature typing* specification given above.

Some typed feature formalisms, notably the ALE formalism, combine type definitions with relation typing specifications into a single definition known as an *appropriateness specification* [Carpenter *et al* 91]. Thus a definition such as:

$$Human = sex : Sex$$

automatically specifies that $sex : Human \longrightarrow Sex$ while at the same time giving a definition for the sort *Human*.

Inheritance

Inheritance is a knowledge base structuring technique to hierarchically organise information. Inheritance based knowledge representation systems have been popularised by knowledge representation languages such as KL-ONE [Brachman & Schmolze 85] which make crucial use of *ako* (*a-kind-of*) and *isa* (*is-a*) *links* to specify inheritance between different type symbols.

For instance, instead of the usual specification of the form:

$$Man = Human \sqcap sex : Male$$

one could separate out the fact that *every Man is a Human being* and instead specify alternatively by:

$$Man \leq Human$$

$$Man = sex : Male$$

The specification $Man \leq Human$ specifies an inheritance link between the sorts Man and $Human$ and denotes the fact that *all men are human.*

One advantage of inheritance specifications is *object orientation* in that the task of specifying a complex knowledge base is broken down into specifying the various objects (*i.e.* sorts) and specifying the inheritance links between the objects. This structuring is beneficial for several purposes. It provides a perspicuous view of the knowledge base allowing for easier management, reuse and modification of the knowledge base as compared to systems (such as PATR-II [Shieber 84]) that do not provide inheritance.

1.2.5 Constraint based Grammars

Current constraint based grammars exemplified by the HPSG grammatical formalism are based around the idea of employing a (possibly extended) typed feature logic for

the representation of grammatical knowledge [Carpenter *et al* 91]. Constraint based grammars follow the philosophy of *grammar as specification* in that they declaratively characterise the different principles of grammar as independent specifications.

Some of the HPSG grammatical principles are given in figure 1.3 which makes use of the following notation.

Notation:

1. We shall use a tuple notation such as:
   $(T_1, \ldots, T_n)$
   to actually mean the feature term:
   $1 : T_1 \sqcap \ldots \sqcap n : T_n$
   where $1, \ldots, n$ are feature labels.

2. We shall use an angle bracket list notation such as:
   $< T_1 | T_n >$
   to actually mean the feature term:
   $cons \sqcap h : T_1 \sqcap t : T_n$
   where *cons* is a undefined sort symbol.

3. We shall use <> to mean the atom *nil*.

Derivation is encoded in HPSG via the *subcategorisation principle* which states the relationship between the *head-dtr* and the *comp-dtrs* which is list valued. Thus HPSG does not need a separate context-free rule schema (*cf.* [Shieber 86]) to govern the derivation. In the definitions given in figure 1.3 the feature labels *tmp1, tmp2* and *tmp3* are employed purely for the purpose of specifying some side conditions that need to be respected.

The *constituent ordering principle* (which is not fully specified in the example in figure 1.3) governs the (list) ordering between the values of the *phon* and the *dtrs* features.

The *head feature principle* is analogous to that in GPSG [Gazdar *et al* 85] and ensures agreement between the *head* and the *mother*.

$$
\begin{aligned}
likes = \quad & phon :< likes' > \sqcap \\
& syn : loc : subcat :< np, np >
\end{aligned}
$$

$$
\begin{aligned}
Subcategorisation\_principle = \quad & syn : \quad loc : subcat : Y \sqcap \\
& dtrs : \quad (head\text{-}dtr : syn : loc : subcat : Z) \sqcap \\
& \qquad comp\text{-}dtrs : dtrs : X) \sqcap \\
& tmp1 : \quad (Append \sqcap (X, Y, Z))
\end{aligned}
$$

$$
\begin{aligned}
Head\_feature\_principle = \quad & syn : loc : head : X \sqcap \\
& dtrs : head\text{-}dtr : syn : loc : head : X
\end{aligned}
$$

$$
\begin{aligned}
Constituent\_order\_principle = \quad & phon : X \sqcap \\
& dtrs : Y \sqcap \\
& tmp3 : Order\_constituents \sqcap (Y, X)
\end{aligned}
$$

$$Append = (<>, X, X)$$

$$
\begin{aligned}
Append = \quad & (< X|R >, Y, < X|Z >) \sqcap \\
& tmp2 : (Append \sqcap (R, Y, Z))
\end{aligned}
$$

$$
\begin{aligned}
Grammatical\_principles = \quad & Subcategorisation\_principle \sqcap \\
& Constituent\_order\_principle \sqcap \\
& Head\_feature\_principle \sqcap \ldots
\end{aligned}
$$

Figure 1.3: HPSG Grammatical Principles

1.2.6 Concept Languages

While typed feature logic based formalisms form an important group of knowledge representation languages for declarative specification of constraint based grammars, frame languages such as KL-ONE form another important group of languages which are employed for a wide range of knowledge representation tasks including natural language applications. Some examples of frame languages are:

1. KL-ONE [Brachman & Schmolze 85]
2. NIKL [Moser 83] [Schmolze 89]
3. KL-TWO [Vilain 85]
4. KANDOR [Patel-Schneider 84]
5. LOOM [MacGregor & Bates 87]
6. BACK [von Luck *et al* 87]

7. KRYPTON [Brachman *et al* 85]
8. CLASSIC [Bordiga *et al* 89]
9. SB-ONE [Kobsa 89]
10. L_{LILOG} [Pletat & von Luck 89]

Such languages are employed not just in general knowledge representation tasks (*see* for instance [Peltason *et al* 91]) but also applications in natural language such as natural language generation using KL-TWO [Nebel & Sondheimer 86], the IDAS generation system [Reiter & Mellish 92], the PENMAN generation system that employs LOOM [Mann & Mathiessen 85] and the IBM LILOG text understanding system [Herzog & Rollinger 91].

In this thesis we shall use the terms *frame languages, concept languages* and *terminological logics* more or less equivalently.

Concept languages can be thought of as the relational analog of feature terms in that they permit relational attributes. They usually come with two constructs for attribute selection, namely *existential membership* (or *existential quantification* over relations) denoted by a term of the form $\exists f : T$ and *universal membership* (or *universal quantification* over relations) denoted by a term of the form $\forall f : T$. Furthermore a propositionally complete concept language allows all the propositional connectives just as in feature logic.

The following informal description characterises the manner in which existential memberships are interpreted.

(11)

formulae		Interpretation
$\exists f : a$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{F } a \end{bmatrix}$
$\exists f : a$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{F } a \\ \text{G } b \end{bmatrix}$
$\exists g : b$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{F } a \\ \text{G } b \end{bmatrix}$
$\exists f : a$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{F } a \\ \text{F } b \end{bmatrix}$
$\exists f : b$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{F } a \\ \text{F } b \end{bmatrix}$
$\exists f : a \sqcap \exists f : b$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{F } a \\ \text{F } b \end{bmatrix}$

As the above interpretation diagram shows, descriptions of the form $\exists f : a \sqcap \exists f : b$ are *consistent* (in contrast to terms of the form $f : a \sqcap f : b$ which is *inconsistent* in feature logic) as long as the attribute f is interpreted relationally and can be given a model in terms of attributive structures that allow relational arcs. By *attributive structures* we mean feature structures that permit multi-valued feature labels.

In constrast to existential memberships, *universal memberships* enforce a different type of well-formedness conditions on the kind of structures that are being modeled. Intuitively speaking, a term of the form $\forall f : T$ requires that every f-value of the attribute structure be in the denotation of the term T. Hence attributive structures of the form given in (12) are *consistent.*

(12)

formulae		Interpretation
$\exists person : \top \sqcap \forall person : male$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{PERSON } male \end{bmatrix}$
$\exists person : \exists name : john \sqcap \forall person : \exists salary : 13$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{PERSON} \begin{bmatrix} \text{NAME } john \\ \text{SALARY } 13 \end{bmatrix} \end{bmatrix}$
$\exists person : \exists name : john \sqcap \forall person : \exists salary : 13$	$\longrightarrow_{\mathcal{I}}$	$\begin{bmatrix} \text{PERSON} \begin{bmatrix} \text{NAME } john \\ \text{SALARY } 13 \end{bmatrix} \\ \text{PERSON} \begin{bmatrix} \text{NAME } mary \\ \text{SALARY } 13 \end{bmatrix} \end{bmatrix}$

The symbol $\top$ (*top*) is a term that conventionally denotes (the collection of) all attributive structures. Similarly the term $\bot$ (*bot*) is conventionally assumed to be *inconsistent*

i.e. denotes the *emptyset.*

Note that the concept term $\forall f : \bot$ is *always* consistent since the empty attributive structure$\left[\ \right]$ is a model. However, the concept term $\forall f : \bot \sqcap \exists f : T$ for any term T is *always* inconsistent.

The usual set-theoretic semantics of feature terms carries over to concept languages too. In other words the denotation of complex concept terms formed out of conjunctions, disjunctions and negations can be given by:

$$[\![S \sqcap T]\!]^{\mathcal{I},\alpha} = [\![S]\!]^{\mathcal{I},\alpha} \cap [\![T]\!]^{\mathcal{I},\alpha}$$

$$[\![S \sqcup T]\!]^{\mathcal{I},\alpha} = [\![S]\!]^{\mathcal{I},\alpha} \cup [\![T]\!]^{\mathcal{I},\alpha}$$

$$[\![\neg S]\!]^{\mathcal{I},\alpha} = \text{all models of } \mathcal{I} - [\![S]\!]^{\mathcal{I},\alpha}$$

This completes an informal introduction to concept languages.

1.3 Thesis Motivation

The are two main sources of motivation for the kind of work being undertaken in this thesis.

1.3.1 Building Integrated Knowledge Representation Systems

Our first source of motivation springs from the desire to devise knowledge representation systems that support both NL applications that are based around feature logic and also other applications (including NL) that employ concept languages of the KL-ONE family. There are several important reasons for desiring this integration. Within computational linguistics there seems to be a division at the moment between NL generation systems that predominantly employ frame languages such as the PENMAN generation system [Mann & Mathiessen 85] and the IDAS generation system [Reiter & Mellish 92] on one hand and NL understanding/translation systems [Zajac 90a] that are based around current constraint based grammars such as HPSG. On the other hand, within the area of artificial intelligence, frame based

languages are arguably the most popular systems for knowledge representation tasks [Peltason *et al* 91] [Owsnicki-Klewe 88] [Peltason 87] etc.

As mentioned earlier, it turns out that a simple minded integration of concept languages with feature terms leads to undecidability. This is due to the fact that while path equations in feature logic does not cause any increase in complexity in feature logic [Smolka 89], the relational analog of path equations known as role-value-maps leads to undecidability in concept languages as shown by [Schmidt-Schauß 89]. This means that a straightforward integration of a variable-free concept language such as $\mathcal{ALC}$ [Schmidt-Schauß & Smolka 91] with variable-free feature terms results leads to undecidability.

Although there are concept languages that permit features and path equations *just* over functional paths such as CLASSIC and the language ALCF formalised in [Hollunder & Nutt 90], they only permit co-reference among functional paths. The fact that they do not provide variables makes it impossible to represent descriptions which need to represent co-reference between the values of arbitrary attribute labels. This makes them unattractive for a large number of applications in computational linguistics wherein precisely this kind of co-reference is desired.

The question we want to address in this thesis is : *Is it possible to devise an integrated feature/concept language that supports variables?*

1.3.2 Better Support for Constraint Based Grammars

Our second source of motivation is to provide enhanced knowledge representation services to constraint based grammars. Current constraint based grammars seem to employ a plethora of as yet unformalised types of descriptive machinery that go beyond what is available in current feature logic based formalisms. We shall in a moment give a sample of such descriptions. What appears to be in common in these descriptions is that they violate the fundamental property of feature logic in that feature labels are to be interpreted as being functional. Instead these descriptions assume that the attribute labels are relational. However, there appear to be several widely differing ways

in which the relational attributes are being interpreted.

The question we want to address in this thesis is : *Is it possible to come up with a general logical framework for dealing with feature logics that are enriched with relational attributes without being biased towards any particular interpretation?*

1.3.3 Some sample uses of relational descriptions

In this section we take a cross-section of examples drawn from the computational linguistics literature which are intended to illustrate the applications of relational descriptions *i.e.* feature descriptions (such as set-valued feature descriptions) which require the interpretation of some of the attributes to be relational.

In [Rounds 88] set-valued feature descriptions were considered as an appropriate vehicle for the modeling of situation theoretic structures [Barwise & Perry 83]. Consider the following card game example adapted from [Barwise 88].

(13) $L \models < Has, john, 3\diamondsuit >$
$L \models < Has, mary, 3\heartsuit >$
$L \models < Sees, john, L >$
$L \models < Sees, mary, L >$

The example in (13) is intended to model a card game situation in which both the players namely *john* and *mary* can *see* the situation denoted by the variable L.

[Rounds 88] provides the following set-valued feature description for the above situation theoretic representation.

(14) $$L = \left\{ \begin{bmatrix} \text{REL} & has \\ \text{PLAYER} & john \\ \text{CARD} & 3\diamondsuit \end{bmatrix}, \begin{bmatrix} \text{REL} & has \\ \text{PLAYER} & mary \\ \text{CARD} & 3\heartsuit \end{bmatrix}, \begin{bmatrix} \text{REL} & sees \\ \text{PLAYER} & john \\ \text{SIT} & L \end{bmatrix}, \begin{bmatrix} \text{REL} & sees \\ \text{PLAYER} & mary \\ \text{SIT} & L \end{bmatrix}, \ldots\ldots\ldots \right\}$$

The description in (14) is intended to model the fact that the situation represented by the variable L supports at least the 4 facts represented as the members of the set-valued description.

Within a concept language like $\mathcal{ALC}$ [Schmidt-Schauß & Smolka 91] augmented with features and variables the above description can be alternatively encoded as given below in (15).

(15)
$$
\begin{aligned}
L \sqcap \; & \exists supports : (rel : has \sqcap \\
& \qquad\qquad player : john \sqcap \\
& \qquad\qquad card : 3\diamondsuit) \sqcap \\
& \exists supports : (rel : has \sqcap \\
& \qquad\qquad player : mary \sqcap \\
& \qquad\qquad card : 3\heartsuit) \sqcap \\
& \exists supports : (rel : sees \sqcap \\
& \qquad\qquad player : john \sqcap \\
& \qquad\qquad sit : L) \sqcap \\
& \exists supports : (rel : sees \sqcap \\
& \qquad\qquad player : mary \sqcap \\
& \qquad\qquad sit : L)
\end{aligned}
$$

The existential memberships (denoted by $\exists supports : \ldots$) are the right kind of descriptions for modeling the partial nature of the set being modeled since it permits the situation modeled by L to support facts other than the 4 facts.

Within languages such as ALCF formalised in [Hollunder & Nutt 90] (which do not support variables) the above description cannot be encoded. Note that this example cannot be encoded by employing path equations just over functional paths.

Our next example (*see* (16) below) of the use of relational attributes is that from [Pollard & Sag 87] (*pp.* 164) where the kind of interpretation attached to relational attributes is completely different from the above examples from [Rounds 88].

(16) **HPSG ADJUNCTS principle**

$$
\begin{bmatrix} \text{HEAD-DTR|SYN|LOC|HEAD|ADJUNCTS } X \\ \text{ADJ-DTRS} \qquad\qquad\qquad\qquad\quad Y \end{bmatrix}
$$

CONDITION $:\forall Y' \in Y \; \exists X' \in X$ such that $SYNTAX(Y')=X'$

Informally speaking the adjuncts principle can be best understood with the aid of a simplified example such as the one given in (17).

(17)
$$
\begin{bmatrix} \text{HEAD-DTR|SYN|LOC|HEAD|ADJUNCTS } \{T_1, \ldots, T_n\} \\ \text{ADJ-DTRS} \qquad\qquad\qquad\qquad\quad \{Y_1, \ldots, Y_n\} \end{bmatrix}
$$

The idea with the adjuncts principle is that each of the Y_i above must satisfy one of the T_j's with the point in mind that there could be more than one distinct Y_i's (say Y_{i_1} and Y_{i_2}) which satisfy the same T_i. This is due to the linguistic fact that adjuncts are iterable.

Within a concept language that supports *universal membership* over relations and features the adjuncts principle can be encoded as given in (18).

(18) $\forall adj-dtrs : syntax : (T_1 \sqcup T_2 \sqcup \ldots \sqcup T_n)$

where *adj-dtrs* is a relational attribute and *syntax* is a functional attribute (*i.e.* a feature label).

This description encodes the fact that *every* value of the attribute *adj-dtrs* must satisfy the description $syntax : (T_1 \sqcup T_2 \sqcup \ldots \sqcup T_n)$, which in turn requires the value of its *syntax* feature to satisfy one of $T_1, T_2 \ldots, T_n$.

Yet another suggested application of set-values is for the modelling of the semantic content of HPSG lexical signs [Pollard & Sag 87] (*pp.* 104). Consider the partial specification of the lexical entry for the verb *see* given in (19).

(19)
$$\left[\begin{array}{l} \text{CONT}\left[\begin{array}{l}\text{REL } \textit{see}\\ \text{SEER } \boxed{2}\\ \text{SEEN } \boxed{1}\end{array}\right] \\ \text{INDS}\left\{\left[\begin{array}{l}\text{VAR } \boxed{1}\\ \text{REST}\left[\begin{array}{l}\text{RELN } \textit{naming}\\ \text{NAME } \textit{sandy}\\ \text{NAMED } \boxed{1}\end{array}\right]\end{array}\right], \left[\begin{array}{l}\text{VAR } \boxed{2}\\ \text{REST}\left[\begin{array}{l}\text{RELN } \textit{naming}\\ \text{NAME } \textit{kim}\\ \text{NAMED } \boxed{2}\end{array}\right]\end{array}\right]\right\} \end{array}\right]$$

The value of the CONT label represents the semantics of the sentence which roughly means $see(kim, sandy)$. The value of the INDS label is a set known as the *semantic indices* which denotes the objects involved in the *seeing* situation. In this case these objects are *Kim* and *Sandy*. The boxed numbers [1] and [2] represent variables and serve as co-reference markers. Semantic indices in HPSG are intended to serve as possible *pronoun antecedents*.

Following [Pollard & Moshier 90] the interpretation of the above kind of HPSG set

descriptions is quite different from the ones we described so far since purely universal and existential quantification over relation symbols alone are insufficient to encode such types of set descriptions. This is so since HPSG set descriptions encode a notion of cardinality of the size of the sets being modeled. It turns out that this kind of set descriptions are closely related to so called *number restrictions* in concept languages such as BACK [von Luck *et al* 87]. The reader is referred to Chapter 3 where we spell out the precise connection.

Finally there is the issue of *linear precedence* constraints. Linear precedence constraints specify in terms of the syntactic/semantic content of *phrases* the relative ordering of words that these phrases contain. Take for instance the HPSG linear precedence constraint [Pollard & Sag 87] (*pp.* 172):

(20) Linear Precedence Constraint 1 (LP1)
HEAD[LEX +] < [ ]

which states the constraint in English that lexical heads precede their sisters.

On the other hand for head final languages such as Japanese and Hindi the above constraint would be stated in reverse order *i.e.*:

(21) Linear Precedence Constraint
[ ] < HEAD[LEX +]

Similarly, the HPSG linear precedence constraint [Pollard & Sag 87] (*pp.* 181) given below:

(22) Linear Precedence Constraint 2 (LP2)
COMPLEMENT[MAJ ¬ V] < [LEX −]

is intended to ensure that NP, PP and AP complements precede other verbal complements and adjuncts.

However what is not clear from the above examples is the mechanism by which linear precedence constraints as stated in the above examples can be encoded within a typed

feature logic based formalism. This is not to say that such an encoding is not possible. For instance, [Engelkamp *et al* 92] provides a mechanism for encoding German word order in the STUF formalism [Dörre & Seiffert 91]. However one disadvantage of such approaches in general is that it often makes the grammar less perspicuous to someone other than the grammar writer himself/herself. What would be desirable is a declarative semantics for linear precedence constraints and a high-level enough language to be able to state them. In addition we would want to extend the constraint solving machinery available in feature logic to deal with linear precedence constraints.

This discussion motivates the kind of logical and computational machinery we investigate in Chapter 4. Our approach is similar in spirit to [Reape 89] which provides a declarative view of linear precedence. However our intention is not just a declarative characterisation but also a computational one.

Conclusions

In the above sections we investigated various kinds of descriptions that are employed within current constraint based grammars that do not have an adequate logical and computational characterisation. In this thesis our aim is to seek a unifying framework within which we can progressively provide both a logical and computational characterisation of diverse kinds of descriptions. Our intention is not to cover each and every kind of descriptive machinery employed within computational linguistics but to build a sufficiently general framework within which it becomes possible to seek formalisation of newer requirements.

1.4 Thesis Summary

In Chapter 2 we address directly the question of whether it is possible to come up with an integrated concept/feature logic that provides support for variables without losing decidability. We show that this is indeed possible by showing that the addition of feature symbols and unquantified variables to the language $\mathcal{ALC}$ [Schmidt-Schauß & Smolka 91], which can be thought as the relational counterpart

of the feature term language [Smolka 88], does not render the logic undecidable. Our logic dubbed $\mathcal{ALV}$ (an acronym for *Attributive Logic with Variables*) is a term language that provides generic support for features, relations, existential and universal quantification over relation symbols and importantly variables. This we believe is a major step towards an integrated typed feature formalism and a generic knowledge representation language. From a KL-ONE perspective $\mathcal{ALV}$ provides enhanced knowledge representation facilities since it provides support for variables thus permitting the representation of complex information structures that are difficult (if not impossible) to represent in most decidable fragments of KL-ONE type knowledge representation systems such as CLASSIC [Bordiga *et al* 89]. From a typed feature logic based perspective, the language $\mathcal{ALV}$ shows a growth path by which typed feature logic based formalisms can be extended so as to support not just NL applications but broad spectrum knowledge representation needs in a single formalism.

In Chapter 3 we address the issue of set-values exclusively. Their integration into feature logic has largely been uncharted territory. Our approach builds on the earlier work on the language $\mathcal{ALV}$. We demonstrate that the HPSG notion of set-values is closely related to so called *number restrictions* used in concept languages such as BACK [von Luck *et al* 87] and formalised in [Hollunder & Nutt 90]. We establish a precise connection between the HPSG notion of set-values and concept languages augmented with number restrictions. This connection has the benefit of establishing an interpretation for the negation of set descriptions, something that has not been investigated within computational linguistics. However instead of introducing *number restrictions* to the language $\mathcal{ALV}$, a seemingly obvious approach, we provide a new type of construct for set descriptions and develop our constraint solving machinery around it. We show that this has the benefit of a more compact representation than that offered by number restrictions which involves using a large number of disjunctions. Our logic $\mathcal{ALS}$ extends $\mathcal{ALV}$ with set descriptions.

The language $\mathcal{ALS}$ does not support set operations such as union, intersection and subset. This deficiency is partly addressed in Appendix A which provide an outline of the semantics and constraints solving rules for augmenting a restricted sublanguage of

$\mathcal{ALS}$ with *union, intersection, subset* and *disjoint union* operations.

In Chapter 4 we address the issue of *linear precedence constraints* in a general setting. We provide a precise semantics for linear precedence constraints as a special case of *strict order relations* [Partee *et al* 90] (*i.e.* irreflexive transitive relations). Our language $\mathcal{ALO}$ extends the positive fragment of the language $\mathcal{ALS}$ (*i.e.* excluding *negations* and *disjunctions*) developed in the previous Chapter with two additions, namely transitive relations and a new construct for expressing linear precedence. The approach makes it possible to express both linear orders and branching orders in a uniform manner.

In Chapter 5 we consider the design of a constraint language dubbed CL-ONE designed expressly to take advantage of the theoretical work undertaken in the previous Chapters. The language CL-ONE turns a concept language such as $\mathcal{ALO}$ into a practical formalism. We study the various choices for adding *type definitions, distributed disjunctions, boolean sort expressions, finite domain constraints, relation typing specifications etc.* We extend the constraint solving machinery available in $\mathcal{ALO}$ to cover these extensions.

In Chapter 6 we illustrate some applications for CL-ONE and recommend some extensions.

We conclude in Chapter 7 by summing up our achievements, discussing some of the shortcomings and suggesting directions for further work.

Chapter 2

Integrating Feature Logics and Concept Languages

2.1 Introduction

In this chapter our aim is to provide an integrated logical and computational account of both feature logics and concept languages. More specifically, our aim is to extend feature logic [Smolka 89] with relations and hence in the process integrate feature logic with the concept language $\mathcal{ALC}$ [Schmidt-Schauß & Smolka 91] which can be thought of as the relational counterpart of feature logic that accommodates *variable-free* concept descriptions with unions, intersections and complements. The lack of variables in concept languages means that certain concepts cannot be expressed in these languages. This makes current concept languages unattractive for a majority of natural language applications such as constraint based grammars wherein variable co-references play a central role for expressing the desired constraints.

One alternative to the use of variables in constraint based grammars that employ feature logics is the use of *path equations*. However, one of the central reasons why the integration of feature logics and concept languages is a difficult enterprise stems from the fact that whereas *path equations* do not give to rise to additional computational complexity in feature logic, their counterparts which are known as *role-value maps* render consistency testing in concept languages *undecidable* [Schmidt-Schauß 89]. Thus integrating concept languages with variable-free feature terms containing path equati-

ons leads to undecidability. The question that is left open is whether adding variables and features to the language $\mathcal{ALC}$ (hence providing virtually all the descriptive machinery of feature logic) makes consistency testing in $\mathcal{ALC}$ *undecidable.* This question is part of a more general question : To what extent can we integrate feature logics and concept languages without sacrificing decidability? We believe that the work in this chapter provides an important step towards answering this question.

In this chapter we provide a brighter picture for this enterprise by showing that adding features and unquantified variables to the language $\mathcal{ALC}$ does not make the logic undecidable. We shall dub our logic $\mathcal{ALV}$ - an acronym for *Attributive Logic with Variables.*

The methodology that we employ for defining $\mathcal{ALV}$ and for developing the consistency testing algorithms for $\mathcal{ALV}$ combines the methods used in
[Schmidt-Schauß & Smolka 91] and [Smolka 89] and owes a lot to their work.

We begin by defining a class of algebras known as *relational algebras* which extend the *feature algebras* that are used in feature logics. *Relational Algebras* extend feature algebras by adding the possibility of relational arcs. Features are construed as binary relations that behave functionally. On the other hand relations are construed as just binary relations. We claim that our definition of relational algebras permits a uniform presentation of the information content of directed graphs that contain edges labelled with relations. We then show that *relational graph algebras* which are the data structure representations of relational algebras provide canonical interpretations for the constraint language $\mathcal{L}_1$. This can be thought of as an extension of Smolka's feature clauses [Smolka 88, Smolka 89] since it provides constraints for describing both functional and relational arcs. It turns out that our definition of relational algebras provides the appropriate kind of interpretation structures when we consider set descriptions in Chapter 3.

We next provide an invariant, complete and terminating set of rewrite rules for the language $\mathcal{L}_1$ that provide consistency testing of any given system of $\mathcal{L}_1$ constraints.

We then define the term description language $\mathcal{ALV}$ that provides unions, intersections,

complements and importantly variables. $\mathcal{ALV}$ extends $\mathcal{ALC}$ by providing variables while it extends feature logic by adding generic support for relations. The constraint solving machinery that we develop for $\mathcal{ALV}$ incorporates the central ideas behind the constraint solving machinery for the language $\mathcal{ALC}$ and that for feature logic.

We then prove the central results of this chapter that our constraint solving rules are invariant, complete and terminating. Our termination claim relies on the assumption that some of the constraint solving rules have higher priority and hence must be activated whenever possible. For this reason the termination proof proves to be a tedious and difficult exercise.

2.2 Relational Algebras and Relational Graph Algebras

In this section we discuss relational algebras and their data structure representations which we call relational graphs. Both of these can be thought of as the relational counterparts of *feature algebras* and *feature graph algebras* developed in [Smolka 89].

For this purpose, we shall fix an alphabet consisting of $f, g, \ldots \in \mathcal{F}$ the set of *relation symbols* and $a, b, \ldots \in \mathcal{At}$ the set of *constant symbols.*

A *relational algebra* $\mathcal{I}$ over the pair $(\mathcal{At}, \mathcal{F})$ is a structure $< \mathcal{U}^I, .^I >$ where:

1. $\mathcal{U}^I$ is a *non-empty* set known as the *universe* of $\mathcal{I}$.

2. $.^I$ is an interpretation function that maps every constant a in $\mathcal{At}$ to an element $a^I \in \mathcal{U}^I$ and every relation $f \in \mathcal{F}$ to a binary relation $f^I \subseteq \mathcal{U}^I \times \mathcal{U}^I$.

   Notation:

   (a) Let $f^I(e)$ denote the set $\{e' \mid (e, e') \in f^I\}$

   (b) Let $f^I(e) \uparrow$ mean $f^I(e) = \emptyset$

   (c) Let $f^I(e) \downarrow$ mean $f^I(e) \neq \emptyset$

   Let $\mid f^I(e) \mid$ denote the cardinality of the set $f^I(e)$.

3. $\mathcal{I}$ is required to satisfy the following properties:

(a) if $a_1 \not\equiv a_2$ then $a_1^I \neq a_2^I$

(b) for any atom $a \in \mathcal{A}t$ and for any relation $f \in \mathcal{F}$ there exists no $e \in \mathcal{U}^I$ such that $(a^I, e) \in f^I$ *i.e.* $f^I(a^I) \uparrow$

(c) $\mid f^I(e) \mid$ is finite

Given some set of nodes and labels, a *relational graph* is a pair (x_0, G) where G is a finite, connected and directed graph whose edges are labelled with relation symbols and x_0 is a node of G known as the *root.* Each edge in G is labelled with a relation symbol f. We write xfy to mean the edge that connects x to y labelled with f. For any relation f and node x a relational graph must satisfy the following criteria:

1. *either* there are no f edges coming out of x *i.e.* f *is undefined on* x
2. *or* there are a finite number of f edges coming out of x

Relational graphs are the data structure representations for the constraints in the constraint language $\mathcal{L}_1$ which we develop in the next section.

We next define *relational graph algebras* which are algebras formed out of relational graphs. Relational graph algebras turn out to be useful since we show in section 2.3.1 that they provide canonical interpretations for constraints in the language $\mathcal{L}_1$.

Let $x, y, \ldots \in \mathcal{V}$ be the set of variables disjoint from $\mathcal{A}t$.

A *relational graph algebra* $\mathcal{R} = < \mathcal{U}^R, .^R >$ over the triple $(\mathcal{V}, \mathcal{A}t, \mathcal{F})$ is constructed as follows:

1. $\mathcal{U}^R$ is the set of all relational graphs formed by drawing the nodes from $\mathcal{V} \cup \mathcal{A}t$ and drawing the edge labels from $\mathcal{F}$ such that:
   - for every $a \in \mathcal{A}t$ if (a, G) is the relational graph with a as its root then $G = \emptyset$.
   - if $(x, G) \in \mathcal{U}^R$ and x' is a node in G such that G' is the subgraph of G with x' as its root then $(x', G') \in \mathcal{U}^R$
2. $((x_0, G), (x_0', G')) \in f^R$ if:

(a) $x_0 f x_0' \in G$ *i.e. an* f*-edge connects* x_0 *to* x_0' in G

(b) G' is the subgraph of G with x_0' as its root

From this definition it is clear that $\mathcal{R}$ is a relational algebra with $\mathcal{U}^R$ as its universe. The relational graph algebra is parameterised over the symbols $(\mathcal{V}, \mathcal{A}t, \mathcal{F})$ used for defining the nodes, atoms and relation symbols.

2.2.1 Information Orderings

Objects in a given relational algebra are ordered by the amount of information that they carry. To quantify precisely the information content of relational algebra objects, in the following we define a pre-ordering known as the *subsumption ordering* on the objects of a given relational algebra.

Definition 2.2.1 [Partial Homomorphism] *Let* $\mathcal{I}$ *and* $\mathcal{J}$ *be two relational algebras. Then a* ***partial homomorphism*** *from* $\mathcal{I}$ *to* $\mathcal{J}$ *is a partial function* $\gamma : \mathcal{U}^I \rightharpoonup \mathcal{U}^J$ *such that:*

1. *if* a *is an atom and* $\gamma(a^I) \downarrow$ *then* $\gamma(a^I) = a^J$.
2. *if* γ *is defined on* e *and* f^I *is defined on* e *then for each* e_i *such that* $(e, e_i) \in f^I$: $\gamma(e_i) \downarrow$ *and* $(\gamma(e), \gamma(e_i)) \in f^J$

A *partial endomorphism* is a partial homomorphism $\gamma : \mathcal{U}^I \rightharpoonup \mathcal{U}^I$.

Definition 2.2.2 [Subsumption Preorder] *We say that an element* e *subsumes (i.e. is more general than) an element* e' *relative to an interpretation* $\mathcal{I}$ *written* $e \preceq^I e'$ *if there exists a partial endomorphism* $\gamma : \mathcal{U}^I \rightharpoonup \mathcal{U}^I$ *such that* $\gamma(e) = e'$.

Our definitions for *partial homomorphism* and *subsumption preorder* conservatively extend the corresponding definitions for feature logic provided in [Smolka 89] by allowing algebras to contain not just graphs with functional edges as is the case with Smolka's *feature logic* but also relational edges. However, importantly our definitions reduce to the corresponding ones for feature logic if we restrict our attention to only feature algebras. Note that every feature algebra is also a relational algebra.

The notion of partial endomorphism and the related notion of subsumption ordering proves to be instrumental when we provide in section 2.3.1 a method for constructing *canonical interpretations.*

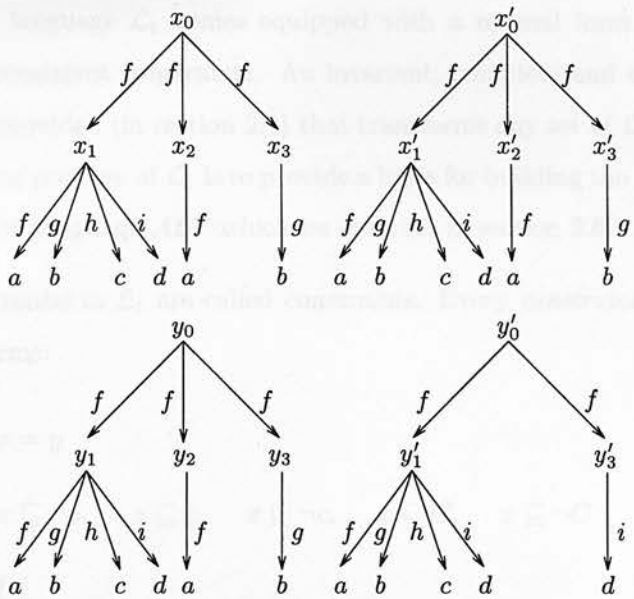


Figure 2.4: Weak equivalence between relational graphs

Definition 2.2.3 [Weak equivalence] *We say that two elements $e_1, e_2 \in \mathcal{U}^I$ are weakly equivalent written $e_1 \simeq^I_w e_2$ if both $e_1 \preceq^I e_2$ and $e_2 \preceq^I e_1$.*

Definition 2.2.4 [Equivalence] *We say that two elements $e_1, e_2 \in \mathcal{U}^I$ are equivalent written $e_1 \simeq^I e_2$ if there exist partial endomorphisms γ and γ^{-1} such that*

- *$\gamma(e_1) = e_2$ and*
- *for every $e \in \mathcal{U}^I$ if $\gamma(e) \downarrow$ then $\gamma^{-1}(\gamma(e)) = e$.*

In this case, we shall refer to γ as a partial isomorphism and γ^{-1} as the inverse of γ.

While it is clear that if e_1 and e_2 are *equivalent* (*e.g.* x_0 and x'_0 in figure 2.4) then they are *weakly equivalent*, the converse does not hold. To see that this is the case it is enough to consider the elements y_0 and y'_0 in figure 2.4 which are weakly equivalent but not equivalent.

2.3 The Language $\mathcal{L}_1$

In this section we introduce the constraint language $\mathcal{L}_1$ that can be considered a generalisation of Smolka's feature clauses [Smolka 88]. $\mathcal{L}_1$ augments feature clauses by providing constraints for describing both functional and relational constraints between

variables. The language $\mathcal{L}_1$ comes equipped with a normal form that allows easy checking for inconsistent constraints. An invariant, complete and terminating set of rewrite rules is provided (in section 2.4) that transforms any set of $\mathcal{L}_1$ constraints into normal form. The purpose of $\mathcal{L}_1$ is to provide a basis for building the constraint solving machinery for the language $\mathcal{ALV}$ which we describe in section 2.6.

Well-formed formulas in $\mathcal{L}_1$ are called *constraints.* Every *constraint* in $\mathcal{L}_1$ has one of the following forms:

$$x \neq y, \quad x = y$$

$$x \dot{\sqsubseteq} a, \quad x \dot{\sqsubseteq} \neg a, \quad x \dot{\sqsubseteq} c, \quad x \dot{\sqsubseteq} \neg c, \quad x \dot{\sqsubseteq} C, \quad x \dot{\sqsubseteq} \neg C$$

$$xfy, \quad xf\uparrow$$

$$x \; \exists f \; y, \quad x \; \forall f \; y$$

where as before $x, y, z \in \mathcal{V}$ the set of *variables*; $f, g \in \mathcal{F}$ the set of *relation symbols*; $a \in \mathcal{A}t$ the set of *atomic symbols* and $C \in \mathcal{P}$ the set of *primitive concept symbols.* In addition we assume that $c \in \mathcal{C}$ is the set of *constant symbols.* We shall further assume that the set of variables $\mathcal{V}$, the set of atoms $\mathcal{A}t$ and the set of constants $\mathcal{C}$ are pairwise distinct.

A *constraint system* C_s is a *finite set* of constraints as defined above. Although $\mathcal{L}_1$ does not provide *disjunctions* the language can be extended with either *distributed disjunctions* [Dörre & Eisele 90] or disjunctions of constraint systems [Smolka 89].

We say that a constraint is *satisfiable* if there exists a relational algebra $\mathcal{I}$ and an $\mathcal{I}$-assignment α that maps:

- every element in $\mathcal{V} \cup \mathcal{C}$ to an element in $\mathcal{U}^I$ such that:
  for distinct constants $c_1, c_2 : \alpha(c_1) \neq \alpha(c_2)$
- every primitive concept symbol C to a subset $\alpha(C) \subseteq \mathcal{U}^I$

such that the following conditions are satisfied:

1. $\mathcal{I}, \alpha \models x = y \iff \alpha(x) = \alpha(y)$
2. $\mathcal{I}, \alpha \models x \neq y \iff \alpha(x) \neq \alpha(y)$
3. $\mathcal{I}, \alpha \models x \doteq a \iff \alpha(x) = a^I$
4. $\mathcal{I}, \alpha \models x \doteq \neg a \iff \alpha(x) \neq a^I$
5. $\mathcal{I}, \alpha \models x \doteq c \iff \alpha(x) = \alpha(c)$
6. $\mathcal{I}, \alpha \models x \doteq \neg c \iff \alpha(x) \neq \alpha(c)$
7. $\mathcal{I}, \alpha \models x \doteq C \iff \alpha(x) \in \alpha(C)$
8. $\mathcal{I}, \alpha \models x \doteq \neg C \iff \alpha(x) \notin \alpha(C)$
9. $\mathcal{I}, \alpha \models xfy \iff f^I(\alpha(x)) = \{\alpha(y)\}$
10. $\mathcal{I}, \alpha \models xf\uparrow \iff f^I(\alpha(x))\uparrow$
11. $\mathcal{I}, \alpha \models x\ \exists f\ y \iff (\alpha(x), \alpha(y)) \in f^I$
12. $\mathcal{I}, \alpha \models x\ \forall f\ y \iff$ Either $f^I(\alpha(x))\uparrow$ or $f^I(\alpha(x)) = \{\alpha(y)\}$

Our slightly unusual notation for constraints of the form $x \doteq c$, $x \doteq C$ *etc.* will alleviate the need for different types of syntax for equivalent constraints when we consider the constraint solving machinery for the term language $\mathcal{ALV}$ in section 2.7.2.

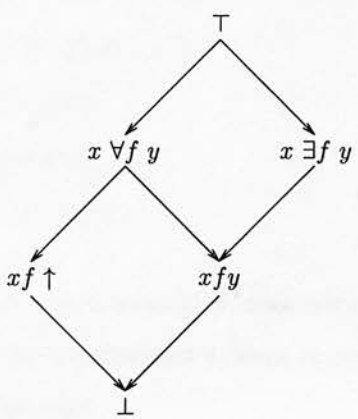


Figure 2.5: Lattice ordering of constraints

The constraint $x\ \forall f\ y$ is a compact representation for the disjunction of the constraints xfy and $xf\uparrow$ and codes up a limited amount of disjunction locally. Since for any relational algebra $\mathcal{I}$ and an assignment α, we have:

$$\mathcal{I},\alpha \models x\ \forall f\ y \Longleftrightarrow \mathcal{I},\alpha \models xfy \ \vee\ \mathcal{I} \models xf\uparrow$$

$$\mathcal{I},\alpha \models x\ \exists\ f\ y \wedge \mathcal{I},\alpha \models x\ \forall f\ y \Longleftrightarrow \mathcal{I},\alpha \models xf\ y$$

$$\neg(\mathcal{I},\alpha \models x\ \exists f\ y \wedge \mathcal{I},\alpha \models xf\uparrow)$$

the constraints xfy, $xf\uparrow$, $x\ \exists f\ y$ and $x\ \forall f\ y$ can be thought as being ordered in the lattice hierarchy as depicted in Figure 2.5 where the symbols $\top$ and $\bot$ stand for the lattice *top* and *bottom* respectively.

Definition 2.3.1 [Consistency] *A constraint system C_s is consistent if there exists a relational algebra $\mathcal{I}$ and an assignment α that satisfies each constraint in C_s. In this case we write $\mathcal{I},\alpha \models C_s$.*

2.3.1 Normal form and canonical interpretations

Definition 2.3.2 [Normal Form] *We say that a constraint system C_s is in normal form if C_s satisfies the following conditions:*

1. *if the constraint $x = y \in C_s$ such that $x \not\equiv y$ then y occurs exactly once in C_s*
2. *if $x \dot{\sqsubseteq} a \in C_s$ and $y \dot{\sqsubseteq} a \in C_s$ then $x \equiv y$*
3. *if $x \dot{\sqsubseteq} c \in C_s$ and $y \dot{\sqsubseteq} c \in C_s$ then $x \equiv y$*
4. *if $xfy \in C_s$ and $xfz \in C_s$ then $y \equiv z$*
5. *if $xfy \in C_s$ then $x\ \exists f\ z \notin C_s$*
6. *if $xfy \in C_s$ then $x\ \forall f\ z \notin C_s$*
7. *if $x\ \exists f\ y \in C_s$ then $x\ \forall f\ z \notin C_s$*

Notice that a constraint system in normal form does not exhibit inconsistency directly. In order to determine whether a constraint system in normal form is inconsistent we define the following notion of *clash*.

We say that a variable x contains a **clash** in a constraint system C_s if either of the following conditions hold:

1. $x \neq x \in C_s$
2. both $x \dot{\sqsubseteq} a_1 \in C_s$ and $x \dot{\sqsubseteq} a_2 \in C_s$ such that $a_1 \not\equiv a_2$
3. both $x \dot{\sqsubseteq} c_1 \in C_s$ and $x \dot{\sqsubseteq} c_2 \in C_s$ such that $c_1 \not\equiv c_2$
4. both $x \dot{\sqsubseteq} \overline{C} \in C_s$ and $x \dot{\sqsubseteq} \neg\overline{C} \in C_s$
   where $\overline{C}$ ranges over a, c, C.
5. both $xfy \in C_s$ and $x \dot{\sqsubseteq} a \in C_s$
6. both $x\ \exists f\ y \in C_s$ and $x \dot{\sqsubseteq} a \in C_s$
7. both $xfy \in C_s$ and $xf \uparrow \in C_s$
8. both $x\ \exists f\ y \in C_s$ and $xf \uparrow \in C_s$

If a variable x contains a clash then it is clear that there is no relational algebra and assignment function that will satisfy all the x-constraints.

Given a relational algebra $\mathcal{I}$, we say that an $\mathcal{I}$-assignment α is a **solution** to a constraint system C_s if $\mathcal{I}, \alpha \models C_s$.

Next, we show that constraint systems in normal form have a special property, namely that we can easily construct a canonical solution for such constraint systems. The following definitions lead to the normal form theorem.

Let $\rightarrow$ be the binary relation between variables in C_s given by $x \rightarrow y$ if $xfy \in C_s$ or $x\ \exists f\ y \in C_s$. Let $\underset{*}{\rightarrow}$ denote the *reflexive* and *transitive* closure of $\rightarrow$.

Definition 2.3.3 [Connected Components] *Given a constraint system C_s in normal form the connected components of a variable x are given by:*

$$con(x, C_s) = \{k \in C_s \mid k \in \{y \dot{\sqsubseteq} c, yfz, y\ \exists f\ z\} \text{ and } x \underset{*}{\rightarrow} y\}$$

Definition 2.3.4 [Canonical Solution] *Let $\mathcal{I}$ be a relational algebra and a C_s be any constraint system. We say that an $\mathcal{I}$-assignment α is a canonical solution of C_s if firstly, α is a solution to C_s and secondly, for any other solution β the following conditions hold:*

- *for every variable x in C_s : $\alpha(x) \preceq^I \beta(x)$*

- *for every constant c in C_s : $\alpha(c) \preceq^I \beta(c)$*
- *for every primitive concept symbol C in C_s :*
  $\alpha(C) = \{\alpha(x) \mid x \dot{\sqsubseteq} C \in C_s\}$

Theorem 2.3.5 [Normal Form theorem] *If C_s is a constraint system in normal form then:*

1. *C_s is inconsistent if there exists a variable $x \in \mathcal{V}(C_s)$ containing a clash*
2. *if C_s contains no variable containing a clash then α is the canonical solution for every $x \in \mathcal{V}(C_s)$ in the relational graph algebra $\mathcal{R}$ if:*
   - *for every variable x in C_s:*
     $\alpha(x) = (\xi(x), RG[con(x, C_s)])$ where
     (a) ξ is an auxiliary function $\xi : \mathcal{V} \longrightarrow \mathcal{V} \cup \mathcal{C}$ defined by:
     - *For each x such that $x \dot{\sqsubseteq} c \in C_s$ let $\xi(x) = c$.*
     - *For each x such that $x \dot{\sqsubseteq} c \notin C_s$ let $\xi(x) = x$.*

     (b) and $RG[C_G]$ is a function that generates a graph from C_G defined by:
     $RG[C_G] = C'_s$ where C'_s is obtained from C_G by:
     - *removing each constraint of the form $x \dot{\sqsubseteq} c$ and instead replacing every occurrence of x with c*
     - *removing each constraint of the form $x\ \exists f\ y$ and replacing it by xfy.*
   - *for every constant symbol c in C_s:*
     - *$\alpha(c) = \alpha(x)$ if $x \dot{\sqsubseteq} c \in C_s$*
     - *$\alpha(c) = (c, \emptyset)$ if $x \dot{\sqsubseteq} c \notin C_s$*

     Note that from the normal form definition given in 2.3.2, for every c in C_s, there is at most one $x \dot{\sqsubseteq} c \in C_s$.
   - *$\alpha(a) = (a, \emptyset)$*
   - *for every primitive concept symbol C in C_s :*
     $\alpha(C) = \{\alpha(x) \mid x \dot{\sqsubseteq} C \in C_s\}$

Proof: For the first part, assume that there exists a variable $x \in \mathcal{V}_s$ containing a clash. Then we know from the satisfiability conditions in definition 2.3.1 that there is no relational algebra that satisfy C_s.

For the second part, we first need to show that $\alpha(x)$ is a solution for every $x \in \mathcal{V}(C_s)$. Let x be any variable in C_s.

1. if $x \dot{\sqsubseteq} c \in C_s$ then we know that $\xi(x) = c$ and $\alpha(x) = (c, RG[con(x, C_s)])$.
2. if $x \dot{\sqsubseteq} C \in C_s$ then we know that $\alpha(x) \in \alpha(C)$ by the definition of $\alpha(C)$.

3. if $x \dot{\sqsubseteq} a \in C_s$ then we know from the definition 2.3.2 that $xfy \notin C_s$ and $x\ \exists f\ y \notin C_s$ and since $\alpha(x) = (a, \emptyset)$, it follows that $\mathcal{R}, \alpha \models x \dot{\sqsubseteq} a$.

4. if $x\ f\ y \in C_s$ then we know that $\alpha(x)$ is the graph $(\xi(x), RG[con(x, C_s)])$ such that $\xi(x) f \xi(y)$ is an edge in $RG[con(x, C_s)]$. Then since $\alpha(y) = (\xi(y), RG[con(y, C_s)])$ and $con(y, C_s) \subseteq con(x, C_s)$, it follows that $f^R(\alpha(x)) = \{\alpha(y)\}$.

5. if $x\ \exists f\ y \in C_s$ then we know that $\alpha(x)$ is the graph $(\xi(x), RG[con(x, C_s)])$ such that $\xi(x)\ \exists f\ \xi(y)$ is an edge in $RG[con(x, C_s)]$. Then since: $\alpha(y) = (\xi(y), RG[con(y, C_s)])$ and $con(y, C_s) \subseteq con(x, C_s)$,
   it follows that $RG[con(y, C_s)]$ is the subgraph of $RG[con(y, C_s)]$ with $\xi(y)$ as its root.
   Hence, $(\alpha(x), \alpha(y)) \in f^R(\alpha(x))$.

6. if $x\ \forall f\ y \in C_s$ then we know from the definition 2.3.2 that both $xfy \notin C_s$ and $x\ \exists f\ y \notin C_s$. Since there are no f-edges coming out of $\alpha(x)$, it follows that $\mathcal{R}, \alpha \models x\ \forall f\ y$.

7. if $xf \uparrow \in C_s$ then we know from the definition 2.3.2 that both $xfy \notin C_s$ and $x\ \exists f\ y \notin C_s$. Since there are no f-edges coming out of $\alpha(x)$, it follows that $\mathcal{R}, \alpha \models xf \uparrow$.

8. if $x \dot{\sqsubseteq} \neg c \in C_s$ then we know from the definition 2.3.2 that $x \dot{\sqsubseteq} c \notin C_s$ hence $\xi(x) \neq c$ and hence $\mathcal{R}, \alpha \models x \dot{\sqsubseteq} \neg c$.

9. if $x \dot{\sqsubseteq} \neg C \in C_s$ then we know from the definition 2.3.2 that $x \dot{\sqsubseteq} C \notin C_s$ hence $\alpha(x) \notin \alpha(C)$ and hence $\mathcal{R}, \alpha \models x \dot{\sqsubseteq} \neg C$.

10. if $x \dot{\sqsubseteq} \neg a \in C_s$ then we know from the definition 2.3.2 that $x \dot{\sqsubseteq} a \notin C_s$ hence $\xi(x) \neq a$ and hence $\mathcal{R}, \alpha \models x \dot{\sqsubseteq} \neg a$.

11. if $x \neq y \in C_s$ then we know from the definition 2.3.2 that there is no constant or atom $\overline{c}$ such that $x \dot{\sqsubseteq} \overline{c} \in C_s$ and $y \dot{\sqsubseteq} \overline{c} \in C_s$ which means $\xi(x) \neq \xi(y)$. This means $\alpha(x) \neq \alpha(y)$ and hence $\mathcal{R}, \alpha \models x \neq y$.

This proves that $\mathcal{R}, \alpha \models C_s$.

We now need to show that α is a canonical solution. Assume that β is any other $\mathcal{I}$-assignment such that $\mathcal{R}, \beta \models C_s$.

Let γ be a partial function defined by:

$\gamma(\alpha(x)) = \beta(x)$ for every variable $x \in \mathcal{V}(C_s)$

$\gamma(a^R) = a^R$ for every atom a

We shall demonstrate that γ is indeed a partial endomorphism.

First, we show that γ is properly defined.

To prove this, note that α is a one-to-one mapping from variables to their interpretations constructed by the normal form theorem. Hence γ which maps the α-image of a variable to the β-image of the same variable is functional since β is also a function.

For any x such that $x \doteq a \in C_s$ we know that $\alpha(x) = \beta(x) = (a, \emptyset)$. Hence assuming $\gamma(a^R) = a^R$ retains the functional nature of γ.

Next, we show that γ has the endomorphism property on every $\alpha(x)$ and $\alpha(c)$ as required by the definition of canonical solution.

1. For every $(e_x, e_y) \in f^R$ we know that there exists either $x \; \exists f \; y \in C_s$ or $xfy \in C_s$ such that $e_x = \alpha(x)$ and $e_y = \alpha(y)$.

   By the definition of γ we know that:

   $\gamma(\alpha(x)) = \beta(x)$ and

   $\gamma(\alpha(y)) = \beta(y)$

   Since $\mathcal{R}, \beta \models C_s$ we know that $(\beta(x), \beta(y)) \in f^R$.

   Hence for every $(e_x, e_y) \in f^R : (\gamma(e_x), \gamma(e_y)) \in f^R$ as required by the definition of partial endomorphism given in section 2.2.1.

2. If $\alpha(c) \downarrow$ on some constant c then we know that $x \doteq c \in C_s$, $\alpha(x) = \alpha(c)$ and $\beta(x) = \beta(c)$.

   Hence to show that $\gamma(\alpha(c)) = \beta(c)$ is endomorphic, it is enough to show that $\gamma(\alpha(x)) = \beta(x)$ is endomorphic.

Finally, for any C in C_s we know that $\alpha(C)$ is the set :

$\alpha(C) = \{\alpha(x) \mid x \dot{\sqsubseteq} C \in C_s\}$

as required by the definition of canonical solution.

This completes the proof.

Hence, $(\xi(x), RG[con(x, C_s)])$ is a canonical interpretation in the relational graph algebra $\mathcal{R}$.

Corollary 2.3.6 *1. Any constraint system C_s in normal form is consistent iff it has a solution in the relational graph algebra $\mathcal{R}$.*

2. If C_s is consistent and is in normal form then it has a canonical solution in the relational graph algebra $\mathcal{R}$.

2.4 Constraint Solving in $\mathcal{L}_1$

In this section we present normalisation rules for $\mathcal{L}_1$ constraint systems and show that every constraint system can be transformed into a normal form from which the consistency of the original constraint system can be determined. The normalisation rules form the basis of reasonably efficient constraint solving algorithms for consistency checking. The feature unification algorithm of [Aït-Kaci & Nasr 86] can easily be extended to our case.

Notation:

1. In the following we shall use $\Phi \cup C_s$ to actually mean the set $\Phi \cup (C_s - \Phi)$.
2. We shall use $[x/y]C_s$ to mean the set obtained from C_s by replacing every occurence of y with x.

We shall employ the following normalisation rules:

(Equals) $\{x = y\} \cup C_s \longrightarrow \{x = y\} \cup [x/y]C_s$
if $x \not\equiv y$ and y occurs in C_s

(Const) $\{x \dot{\sqsubseteq} \overline{c}, y \dot{\sqsubseteq} \overline{c}\} \cup C_s \longrightarrow \{x = y, x \dot{\sqsubseteq} \overline{c}\} \cup C_s$
where $\overline{c}$ ranges over a, c.

(Feat) $\{xfy,\ xFz\} \cup C_s \longrightarrow \{xfy,\ y = z\} \cup C_s$
where F ranges over $f, \exists f, \forall f$

(ExForall) $\{x\ \exists f\ y,\ x\ \forall f\ z\} \cup C_s \longrightarrow \{xfy, y = z\} \cup C_s$

Definition 2.4.1 [Equivalence] *A constraint system C_s is equivalent to another constraint system C'_s if for any relational algebra $\mathcal{I}$ and $\mathcal{I}$-assignment α:*

$$\mathcal{I}, \alpha \models C_s \Longleftrightarrow \mathcal{I}, \alpha \models C'_s$$

We next show that our constraint solving rules transform any given constraint system into an equivalent constraint system.

Theorem 2.4.2 [Invariance] *Every normalisation rule transforms any given $\mathcal{L}_1$ constraint system C_s into an equivalent constraint system.*

Proof: Since our normalisation rules do not eliminate existing variables and do not introduce any new ones, to prove the invariance claim it is enough to prove that each of the normalisation rules preserves the interpretation of every existing variable. It is quite easy to see that every normalisation rule preserves the semantics of every existing variable.

Theorem 2.4.3 [Termination] *There is no infinite chain issuing from the application of the normalisation rules.*

Proof: To prove the termination claim we shall use a complexity measure based on multiset orderings due to [Dershowitz & Manna 78] that is sensitive to the degree to which the given constraint system is in normal form. We define the complexity measure as follows:

1. Consider each constraint system C_s as a multiset.

2. A complexity measure on the multiset C_s is a multiset $K(C_s)$ formed by replacing each constraint k in C_s by a natural number $\mid k \mid$ defined as follows:

$$\mid x\ \exists f\ y \mid = \mid x\ \forall f\ y \mid = 4$$

$$\mid xfy \mid = \mid xf\uparrow \mid = 2$$

$$\mid x \dot{\sqsubseteq} a \mid = \mid x \dot{\sqsubseteq} \neg a \mid = \mid x \dot{\sqsubseteq} c \mid = \mid x \dot{\sqsubseteq} \neg c \mid = \mid x \dot{\sqsubseteq} C \mid = \mid x \dot{\sqsubseteq} \neg C \mid = 2$$

$$\mid x \neq y \mid = 1$$

$| x = y | = 1$ if y occurs more than once in C_s

$| x = y | = 0$ otherwise

Define a transitive, irreflexive order $\succ$ on the complexities
(see [Dershowitz & Manna 78]) by:

$K(C_s) \succ K(C'_s)$ if

there exist multisets X, Y, Z where X is non-empty such that
$K(C_s) = X \cup Z$ and $K(C'_s) = Y \cup Z$ and
$(\forall y \in Y)(\exists x \in X)\ x > y$

where the ordering $>$ is the usual *greater than* ordering on the natural numbers.

By definition, the ordering $\succ$ is a well-founded order on the complexities. Since every simplification rule reduces the complexity with respect to this order, we know that the normalisation procedure terminates.

Theorem 2.4.4 [Completeness of Rewriting] *The application of the above normalisation rules to any given constraint system transforms the given constraint system into normal form.*

Proof: To prove the completeness claim, we need to ensure that if a constraint system is not in normal form then one of the normalisation rules apply. For each of the conditions for normal form given in definition 2.3.2, it is easy to see that there is a corresponding rule. Hence, together with the termination theorem we have the above claim.

Corollary 2.4.5

1. *For any constraint system C_s it is decidable whether C_s is consistent.*
2. *Any consistent constraint system C_s has a canonical solution in the relational graph algebra $\mathcal{R}$.*

The above claims follow since we know that any given constraint system can be transformed into an equivalent normal form by using the normalisation rules. From the termination claim we know that this process terminates. Once a constraint system is in normal form we know from the normal form theorem that the (in)consistency

of the constraint system can be determined (- the first claim) and furthermore if the constraint system is consistent then it has a canonical solution in the relational graph algebra $\mathcal{R}$ (- the second claim).

2.5 $\mathcal{L}_1$ and PATR-II

Although we have shown that $\mathcal{L}_1$ is a constraint language that comes fully equipped with an invariant, complete and terminating constraint solving mechanism what we haven't shown is how this relates to existing constraint languages. In this section we shall show that the language $\mathcal{L}_1$ can be easily extended to supercede the PATR-II formalism developed at SRI by *Stuart Shieber* [Shieber 84]. Although our motivation for introducing $\mathcal{L}_1$ is mainly to provide a basis for building constraint solving machinery for the term language $\mathcal{ALV}$ (see section 2.6), this section will show that for applications where the full expressivity of $\mathcal{ALV}$ is not needed the language $\mathcal{L}_1$ can easily be extended to a PATR-II type language.

From an application point of view, the problem with $\mathcal{L}_1$ as described in the preceding sections is that it is quite cumbersome in practical use. This is due to the fact that describing any useful concepts in $\mathcal{L}_1$ involves introducing a large number of variables purely because $\mathcal{L}_1$ can express constraints only between variables.

In the PATR-II formalism one extra device is provided namely *path equations*. A *path equation* is a constraint of the form $xP = yQ$ where P and Q are lists of feature symbols (*i.e.* role symbols that are interpreted as partial functions). Path equations provide a fairly compact representation for expressing useful linguistic constraints in unification-based grammar formalisms [Gazdar & Mellish 89][Shieber 86]. In the following we shall demonstrate how an extended notion of path equation can be accommodated in $\mathcal{L}_1$.

A **path** is a list of role symbols as defined by the following BNF syntax:

$$P \longrightarrow nil \mid f.P \mid \exists f.P \mid \forall f.P$$

where we shall employ *nil* to denote the empty path.

Given a relational algebra $\mathcal{I}$ the interpretation of *paths* is provided by extending the

interpretation provided by $\mathcal{I}$ for each role symbol as follows:

- $nil^I(x) = \{x\}$
- $(f.P)^I(x) = P^I(y)$ where $f^I(x) = \{y\}$
- $(\exists f.P)^I(x) = \bigcup_{y \in f^I(x)} P^I(y)$
- $(\forall f.P)^I(x) \uparrow$ if $f^I(x) \uparrow$
- $(\forall f.P)^I(x) = P^I(y)$ where $f^I(x) = \{y\}$

A **path constraint** is a constraint of the form:

$$xPy, \quad xP.f \uparrow$$

A **path equation** is a constraint of the form:

$$xP = yQ$$

where P and Q are paths.

The interpretation of *path constraints* and *path equations* is provided by the following definitions:

1. $\mathcal{I}, \alpha \models xPy \iff (\alpha(x), \alpha(y)) \in P^I$
2. $\mathcal{I}, \alpha \models xP.f \uparrow \iff (P.f)^I(\alpha(x)) \uparrow$
3. $\mathcal{I}, \alpha \models xP = yQ \iff (P^I(\alpha(x)) \downarrow \wedge Q^I(\alpha(y)) \downarrow) \Rightarrow (P^I(\alpha(x)) \cap Q^I(\alpha(y)) \neq \emptyset)$

By abuse of terminology, we shall call the augmented language containing path constraints and path equations as $\mathcal{L}_1$.

The existing constraint solving machinery for $\mathcal{L}_1$ is extended by adding the following rules:

Notation: In the following we shall use:

(a) F to mean either $f, \exists f$ or $\forall f$

(b) PV to mean $P\ z$ or $P.f \uparrow$

1. $\{x\ F.PV\} \cup C_s \longrightarrow \{xFy,\ y\ PV\} \cup C_s$
   where y is new
2. $\{x\ nil\ y\} \cup C_s \longrightarrow \{x = y\} \cup C_s$
3. $\{xP = yQ\} \cup C_s \longrightarrow \{xPz,\ yQz\} \cup C_s$
   where z is new

Proposition 2.5.1 *Both path constraints and path equations can be eliminated in linear time.*

We shall leave it to the reader to verify that our additional rules are invariant, complete and terminating. However the notion of invariance applicable to the augmented language is a weaker one since eliminating path equations involve introducing new variables as the above normalisation rules demonstrate.

Our path constraints and path equations extend PATR-II path equations by allowing relational arcs (by using $\exists f$) and disjunctive specification of functional arcs (by using $\forall f$). However, from a practical perspective this may not prove greatly advantageous since most natural language applications would require relational arcs in conjunction with set descriptions so that set-membership constraints can be easily expressed. We believe that the full potential of a constraint language like $\mathcal{L}_1$ and path equations can be exploited if we add constraints for describing sets. This is essentially what we shall do in Chapter 3.

2.6 $\mathcal{ALV}$: A Concept Description Language with Variables

Although the constraint language $\mathcal{L}_1$ is a knowledge representation language in its own right, it is not good enough for every natural language application even if we include path equations. This is so because $\mathcal{L}_1$ can be used to describe only single individuals in a domain but not groups of individuals having a certain property. For instance, the concept that defines *the group of individuals who have red hair* cannot be represented in $\mathcal{L}_1$.

In this section we provide a *term description language* $\mathcal{ALV}$ that can be viewed as an *attributive concept description* language in the sense of [Schmidt-Schauß & Smolka 91]. $\mathcal{ALV}$ complements the work in [Schmidt-Schauß & Smolka 91] by adding *features* (*i.e. functional roles*) and *variables* (hence *restricted agreements*) thus bridging the gap between feature logics and concept description languages. Since every construct in the language $\mathcal{ALC}$ [Schmidt-Schauß & Smolka 91] has a $\mathcal{ALV}$ counterpart the language $\mathcal{ALV}$ can be thought of as a superset of $\mathcal{ALC}$.

Terms in $\mathcal{ALV}$ are *set denoting* expressions that denote subsets of the universe $\mathcal{U}^I$. This is as opposed to the element denoting variables available in $\mathcal{L}_1$. In order to test consistency of $\mathcal{ALV}$ terms we first extend the language $\mathcal{L}_1$ to the language $\mathcal{L}_2$ and then we provide a consistency preserving translation between terms in $\mathcal{ALV}$ and constraints in the language $\mathcal{L}_2$ such that a term T in $\mathcal{ALV}$ is consistent *iff* its translation in $\mathcal{L}_2$ is consistent. This method follows the approach taken in [Smolka 88, Schmidt-Schauß 89, Donini *et al* 91, Hollunder & Nutt 90].

Syntax of $\mathcal{ALV}$

A term description in $\mathcal{ALV}$ is given by the following BNF syntax, where the symbols S, T denote terms; $x, y \in \mathcal{V}$ denote variables; $f, g \in \mathcal{F}$ denote relation symbols; $c \in \mathcal{C}$ denotes constant symbols; $a \in \mathcal{At}$ denotes atomic symbols; $C \in \mathcal{P}$ denotes primitive concepts such that $\top, \bot \in \mathcal{P}$:

$$S, T \longrightarrow x \mid a \mid c \mid C \mid \exists f : T \mid \forall f : T \mid S \sqcap T \mid S \sqcup T \mid \neg T$$

Furthermore, we assume that $\mathcal{Fs} \subseteq \mathcal{F}$ is a set of feature symbols which are interpreted as being functional.

Intuitively, every term in $\mathcal{ALV}$ denotes a subset of $\mathcal{U}^I$. The term $\exists f : T$ where $f \in \mathcal{Fs}$ denotes the set of all elements such that f^I is defined, behaves functionally and their f^I-value is in the denotation of T. Similarly, the term $\exists f : T$ where $f \notin \mathcal{Fs}$ denotes the set of all elements such that there is an f^I-value which is in the denotation of T. A rigorous definition of the semantics is provided next.

Semantics of terms in $\mathcal{ALV}$

To provide the interpretation for $\mathcal{ALV}$ terms we define a denotation function $[\![.]\!]^{\mathcal{I},\alpha}$ that given a relational algebra $\mathcal{I}$ and an assignment α maps terms to subsets of $\mathcal{U}^I$. Firstly, we need to further constrain:

I by:

- For every element $e \in \mathcal{U}^I$ and every feature $f \in \mathcal{F}s$:
  if $(e, e') \in f^I$ and $(e, e'') \in f^I$ then $e' \equiv e''$

α by:

- $\alpha(\top) = \mathcal{U}^I$
- $\alpha(\bot) = \emptyset$

The function $[\![.]\!]^{\mathcal{I},\alpha}$ is then defined as follows:

$$[\![x]\!]^{\mathcal{I},\alpha} = \{\alpha(x)\}$$

$$[\![a]\!]^{\mathcal{I},\alpha} = \{a^I\}$$

$$[\![c]\!]^{\mathcal{I},\alpha} = \{\alpha(c)\}$$

$$[\![C]\!]^{\mathcal{I},\alpha} = \alpha(C)$$

$$[\![\exists f : T]\!]^{\mathcal{I},\alpha} = \{e \in \mathcal{U}^I \mid \exists e' : (e, e') \in f^I \ \wedge \ e' \in [\![T]\!]^{\mathcal{I},\alpha}\}$$

$$[\![\forall f : T]\!]^{\mathcal{I},\alpha} = \{e \in \mathcal{U}^I \mid \forall e' : (e, e') \in f^I \ \Rightarrow e' \in [\![T]\!]^{\mathcal{I},\alpha}\}$$

$$[\![S \sqcap T]\!]^{\mathcal{I},\alpha} = [\![S]\!]^{\mathcal{I},\alpha} \cap [\![T]\!]^{\mathcal{I},\alpha}$$

$$[\![S \sqcup T]\!]^{\mathcal{I},\alpha} = [\![S]\!]^{\mathcal{I},\alpha} \cup [\![T]\!]^{\mathcal{I},\alpha}$$

$$[\![\neg T]\!]^{\mathcal{I},\alpha} = \mathcal{U}^I - [\![T]\!]^{\mathcal{I},\alpha}$$

$$[\![\top]\!]^{\mathcal{I},\alpha} = \mathcal{U}^I$$

$$[\![\bot]\!]^{\mathcal{I},\alpha} = \emptyset$$

We say that a term T is **consistent** if there exists an relational algebra $\mathcal{I}$ and a $\mathcal{I}$-assignment α such that $[\![T]\!]^{\mathcal{I},\alpha} \neq \emptyset$.

Theorem 2.6.1 *For any $\mathcal{I}$-interpretation* $[\![\neg\exists f : T]\!]^{\mathcal{I}} = [\![\forall f : \neg T]\!]^{\mathcal{I}}$.

Proof:

$$
\begin{aligned}
&[\![\neg\exists f : T]\!]^{\mathcal{I},\alpha} = \mathcal{U}^I - \{e \in \mathcal{U}^I \mid \exists e' \; s.t \; (e,e') \in f^I \; and \; e' \in [\![T]\!]^{\mathcal{I},\alpha}\} \\
&= \{e \in \mathcal{U}^I \mid \forall e' \; s.t \; (e,e') \in f^I \;\Rightarrow\; e' \notin [\![T]\!]^{\mathcal{I},\alpha}\} \\
&= \{e \in \mathcal{U}^I \mid \forall e' \; s.t \; (e,e') \in f^I \Rightarrow \; e' \in [\![\neg T]\!]^{\mathcal{I},\alpha}\} \\
&= [\![\forall f : \neg T]\!]^{\mathcal{I},\alpha}
\end{aligned}
$$

Hence we have the theorem.

Theorem 2.6.2 *For any $\mathcal{I}$-interpretation* $[\![\neg\forall : T]\!]^{\mathcal{I}} = [\![\exists : \neg T]\!]^{\mathcal{I}}$.

Proof: We can prove the above in an analogous fashion.

These results provide us with a mechanism to simplify negative constraints by effectively allowing negations to be pushed inwards.

Now, we are ready to consider constraint systems for checking consistency of term descriptions in the language $\mathcal{ALV}$.

2.7 Consistency checking in $\mathcal{ALV}$

Our technique for consistency checking of $\mathcal{ALV}$ terms combines the ideas presented in [Smolka 88] and [Schmidt-Schauß & Smolka 91]. Firstly, we augment the constraint language $\mathcal{L}_1$ with **big variables** written as $X, Y, Z, etc..$ *Big variables* are set denoting expressions that denote *non-empty* subsets of the universe $\mathcal{U}^I$ *i.e.*:

$\alpha(X) \subseteq \mathcal{U}^I$ for any *big variable* X and

$[\![X]\!]^{\mathcal{I},\alpha} = \alpha(X)$

Secondly, to enable a consistency preserving translation of every $\mathcal{ALV}$ term description into a constraint system in the augmented language $\mathcal{L}_1$, we extend the usage of $\dot{\sqsubseteq}$ constraints in the following.

Notation: We shall use $\overline{X}$ to range over either x or X.

The syntax of $\mathcal{L}_1$ is extended by adding the following constraints:

$$\overline{X} \dot{\sqsubseteq} T, \quad x \dot{\sqsubseteq} \overline{Y}, \quad \overline{X} \dot{\sqsubseteq} \exists f : \overline{Y}, \quad \overline{X} \dot{\sqsubseteq} \forall f : \overline{Y}$$

We shall refer to all constraints involving $\dot{\sqsubseteq}$ (including $\mathcal{L}_1$ constraints) as **containment constraints.**

The augmented language will be referred to as $\mathcal{L}_2$ which adds *big variables* and extended *containment constraints* to the language $\mathcal{L}_1$.

Notation: Any containment constraint will be compactly represented by $\overline{X} \dot{\sqsubseteq} \Gamma$.

The semantics of containment constraints is given by the following definition.

Definition 2.7.1 *We say that a relational algebra $\mathcal{I}$ and an $\mathcal{I}$-assignment α satisfies a containment constraint $\overline{X} \dot{\sqsubseteq} \Gamma$ iff:*

$$[\![\overline{X}]\!]^{\mathcal{I},\alpha} \subseteq [\![\Gamma]\!]^{\mathcal{I},\alpha}$$

It is easy to see that this definition preserves the semantics of $\mathcal{L}_1$ constraints containing $\dot{\sqsubseteq}$.

Lemma 2.7.2 *A $\mathcal{ALV}$ term T is consistent iff there exists a variable x such that the $\mathcal{L}_2$ constraint system C_s where $C_s = \{x \dot{\sqsubseteq} T\}$ is consistent.*

Proof: Trivial. Since if there exists a relational algebra $\mathcal{I}$ and an assignment α such that $[\![T]\!]^{\mathcal{I},\alpha}$ is non-empty, then one can construct an assignment β such that β is identical to α except that for some variable $x \notin \mathcal{V}(C_s) : \beta(x) \in [\![T]\!]^{\mathcal{I},\alpha}$. Hence, $\mathcal{I}, \beta \models \{x \dot{\sqsubseteq} T\}$. Conversely, if $\mathcal{I} \models \{x \dot{\sqsubseteq} T\}$ then $\{\alpha(x)\} \subseteq [\![T]\!]^{\mathcal{I},\alpha} \neq \emptyset$.

Containment constraints in $\mathcal{L}_2$ are required to obey a global restriction known as the *acyclicity condition.* We shall define this condition in the following.

By slight overloading of terminology we shall define the following path relation between big variables.

Definition 2.7.3 [Path] *Given a constraint system C_s the relation $\underset{P}{\rightarrow}$ that relates two big variables and a path P is inductively defined as the least relation satisfying:*

1. $X \underset{\epsilon}{\rightarrow} X$
2. $X \underset{f}{\rightarrow} Y$ *if* $X \dot{\sqsubseteq} \exists f : Y \in C_s$ *or* $X \dot{\sqsubseteq} \forall f : Y \in C_s$

3. $X \underset{[\sqcup]}{\rightarrow} Y$ *if* $X \dot{\sqsubseteq} Y \sqcup \overline{Z} \in C_s$

4. $X \underset{[\sqcup]}{\rightarrow} Z$ *if* $X \dot{\sqsubseteq} \overline{Y} \sqcup Z \in C_s$

5. $X \underset{P \bullet Q}{\rightarrow} Z$ *if* $X \underset{P}{\rightarrow} Y$ *and* $Y \underset{Q}{\rightarrow} Z$

where the operation $\bullet$ *is the* concatenation *operation over strings.*

Definition 2.7.4 [Height] *The height of a big variable X in C_s is the length of the longest path P that spans X and some other variable Y. The height of a variable x is* 0. *In other words:*

- $height(X) = maximum(\{length(P) \mid X \underset{P}{\rightarrow} Y \text{ in } C_s\})$
- $height(x) = 0$

Corollary 2.7.5 *If any of $X \dot{\sqsubseteq} \exists f : Y \in C_s$, $X \dot{\sqsubseteq} \forall f : Y \in C_s$ or $X \dot{\sqsubseteq} Y \sqcup Z \in C_s$ then $height(X) > height(Y)$.*

Definition 2.7.6 [Acyclicity Condition] *We say that a given $\mathcal{L}_2$ constraint system C_s satisfies the acyclicity condition if for any big variable X occurring in C_s there is no non-empty path P that makes $X \underset{P}{\rightarrow} X$.*

In section 2.9 we show that testing consistency of $\mathcal{L}_2$ constraints containing *path equations* (where paths are restricted to functional chains) and violating the acyclicity condition is in general *undecidable.* However, we shall see that since big variables are not available in $\mathcal{ALV}$ any consistency preserving translation of $\mathcal{ALV}$ terms into a $\mathcal{L}_2$ constraint system obeys the acyclicity condition.

We are now ready to present the normal form and normalisation rules for $\mathcal{L}_2$ constraint systems.

2.7.1 Normal Form

Analogous to the language $\mathcal{L}_1$ the language $\mathcal{L}_2$ too admits a normal form from which (in)consistency of a given constraint system can be easily detected.

Definition 2.7.7 [Normal Form] *We say that a given $\mathcal{L}_2$ constraint system C_s is in normal form if it satisfies the following conditions in addition to the conditions for normal form for the language $\mathcal{L}_1$ given in definition 2.3.2:*

1. *if $X \dot{\sqsubseteq} \Gamma \in C_s$ then Γ must be one of the following forms:*

$$\overline{C}, \exists f : \overline{Y}, \forall f : \overline{Y}$$

where $\overline{C}$ *ranges over* $x, \neg x, C, \neg C, \neg a, \neg c$.
This means that constraints of the form $X \dot{\sqsubseteq} a$ *and* $X \dot{\sqsubseteq} c$ *are not permitted in the normal form.*

2. *if* $x \dot{\sqsubseteq} \Gamma \in C_s$ *then* Γ *must be one of the following forms:*

$$\overline{C}, \overline{Y}, \forall f : \overline{Y}, \overline{Y} \sqcup \overline{Z}$$

where $f \notin \mathcal{F}s$ *and* $\overline{C}$ *ranges over* $C, \neg C, \neg a, \neg c$

3. *if* $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \overline{C} \in C_s$ *then* $x \dot{\sqsubseteq} \overline{C} \in C_s$ *where* $\overline{C}$ *ranges over* $\neg x, C, \neg C$ *and* $\neg c$.

4. *if* $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} y \in C_s$ *then* $x \equiv y$.

5. *if* $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \overline{Y} \sqcup \overline{Z}$ *then either of the following conditions hold:*

   - $x \equiv \overline{Y}$
   - $x \equiv \overline{Z}$
   - $x \dot{\sqsubseteq} \overline{Y} \in C_s$ *and* $\overline{Y} \equiv Y$
   - $x \dot{\sqsubseteq} \overline{Z} \in C_s$ *and* $\overline{Z} \equiv Z$

6. *if* $x \dot{\sqsubseteq} \overline{Y} \sqcup \overline{Z}$ *then either of the following conditions hold:*

   - $x \equiv \overline{Y}$
   - $x \equiv \overline{Z}$
   - $x \dot{\sqsubseteq} \overline{Y} \in C_s$ *and* $\overline{Y} \equiv Y$
   - $x \dot{\sqsubseteq} \overline{Z} \in C_s$ *and* $\overline{Z} \equiv Z$

7. *if* $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \forall f : \overline{Y}$ *and* $xFy \in C_s$ *(where* F *ranges over* $f, \exists f$*) then:*

   - *either* $y \equiv \overline{Y}$ *or*
   - $\overline{Y} \equiv Y$ *and* $y \dot{\sqsubseteq} Y \in C_s$.

8. *if* $x \dot{\sqsubseteq} \forall f : \overline{Y}$ *and* $xFy \in C_s$ *(where* F *ranges over* $f, \exists f$*) then either of the following conditions holds:*

   - $y \equiv \overline{Y}$ *or*
   - $\overline{Y} \equiv Y$ *and* $y \dot{\sqsubseteq} Y \in C_s$.

9. *if* $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \exists f : \overline{Y} \in C_s$ *then either* $xfy \in C_s$ *or* $x \exists f\ y \in C_s$ *such that either of the following conditions hold:*

   - $y \equiv \overline{Y}$ *or*
   - $\overline{Y} \equiv Y$ *and* $y \dot{\sqsubseteq} Y \in C_s$.

Given a $\mathcal{L}_2$ constraint system in normal form we say that a variable $x \in \mathcal{V}(C_s)$ contains a **clash** if either x contains a clash according to the definition of *clash* for $\mathcal{L}_1$ or $x \dot{\sqsubseteq} \bot \in C_s$.

Theorem 2.7.8 [Normal Form] *A $\mathcal{L}_2$ constraint system C_s that is in normal form is consistent* iff *there is no variable $x \in \mathcal{V}(C_s)$ such that x contains a clash.*

Proof: For the first part, assume that there exists variable $x \in \mathcal{V}(C_s)$ such that x contains a clash then we know from the language $\mathcal{L}_1$ that there is no relational algebra and an assignment that satisfies C_s.

For the second part, assume that there is no x such that x contains a clash, then we shall show that C_s has a model in the relational graph algebra $\mathcal{R}$ given by:

1. $\alpha(x) = (\xi(x), RG[con(x, C_s)])$ for every $x \in \mathcal{V}(C_s)$.
2. $a^R = (a, \emptyset)$
3. $\alpha(c) = \alpha(x)$ if $x \dot{\sqsubseteq} c \in C_s$
4. $\alpha(c) = (c, \emptyset)$ if $x \dot{\sqsubseteq} c \notin C_s$
5. $\alpha(C) = \{\alpha(x) \mid x \dot{\sqsubseteq} C \in C_s\}$
6. $\alpha(X) = \{\alpha(x) \mid x \dot{\sqsubseteq} X \in C_s\}$

Note that ξ, con and RG are as defined by the normal form theorem 2.3.5 for the language $\mathcal{L}_1$.

Since α already satisfies all the $\mathcal{L}_1$ constraints (from the normal form theorem 2.3.5), to prove this it is enough to show that for every variable $x \in \mathcal{V}(C_s)$ and every $X \in \mathcal{V}(C_s)$ the following conditions are satisfied:

1. if $x \dot{\sqsubseteq} X \in C_s$ then $\alpha(x) \in \alpha(X)$.
2. if $x \dot{\sqsubseteq} \overline{Y} \sqcup \overline{Z} \in C_s$ then $\alpha(x) \in [\![\overline{Y}]\!]^{\mathcal{R},\alpha}$ or $\alpha(x) \in [\![\overline{Z}]\!]^{\mathcal{R},\alpha}$.
3. if $xFy \in C_s$ and $x \dot{\sqsubseteq} \forall f : \overline{Y} \in C_s$ where F ranges over $f, \exists f$ then $\alpha(y) \in [\![Y]\!]^{\mathcal{R},\alpha}$.
4. for every constraint of the form $X \dot{\sqsubseteq} \Gamma$:
   $\mathcal{R}, \alpha \models X \dot{\sqsubseteq} \Gamma$

We prove the above case by case.

1. if $x \dot{\sqsubseteq} X \in C_s$ then from the definition of $\alpha(X)$ it follows that $\alpha(x) \in \alpha(X)$.

2. if $x \dot{\sqsubseteq} \overline{Y} \sqcup \overline{Z} \in C_s$ then we know that either of the following hold:

   (a) $x \equiv \overline{Y}$

   (b) $x \equiv \overline{Z}$

   (c) $x \equiv \overline{Z}$

   (d) $x \dot{\sqsubseteq} \overline{Y} \in C_s$

   (e) $x \dot{\sqsubseteq} \overline{Z} \in C_s$

   Hence if $\alpha(x)$ satisfies the constraint $x \dot{\sqsubseteq} \overline{Y}$ or the constraint $x \dot{\sqsubseteq} \overline{Z}$ then $\alpha(x)$ satisfies the constraints $x \dot{\sqsubseteq} \overline{Y} \sqcup \overline{Z}$.

3. if xFy, $X \dot{\sqsubseteq} \forall f : \overline{Y} \in C_s$ (where F ranges over f and $\exists f$) then we know from the normal form theorem that $y \equiv \overline{Y}$ or $y \dot{\sqsubseteq} \overline{Y} \in C_s$ hence if $\alpha(x)$ satisfies the constraints $y \dot{\sqsubseteq} \overline{Y}$ for each y then $\alpha(x)$ satisfies the constraint $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \forall f : \overline{Y}$.

4. We prove that $\mathcal{R}, \alpha \models X \dot{\sqsubseteq} \Gamma$ for every $X \dot{\sqsubseteq} \Gamma \in C_s$ by induction on the height of X.

   Base Cases

   (a) if $X \dot{\sqsubseteq} \Gamma \in C_s$ such that there is no variable x such that $x \dot{\sqsubseteq} X \in C_s$ then by our construction $\alpha(X) = \emptyset$. Hence $\mathcal{R}, \alpha \models X \dot{\sqsubseteq} \Gamma$.

   (b) if $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \overline{C} \in C_s$ where $\overline{C}$ ranges over x, a, c, C then we know that $X \dot{\sqsubseteq} \neg\overline{C} \notin C_s$ hence $\alpha(x)$ satisfies the constraints $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \overline{C}$ by construction.

   (c) if $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \neg\overline{C} \in C_s$ where $\overline{C}$ ranges over x, c, C then we know that $X \dot{\sqsubseteq} \overline{C} \notin C_s$ hence $\alpha(x)$ satisfies the constraints $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \neg\overline{C}$ by construction.

   (d) if $x \dot{\sqsubseteq} X$, $X \dot{\sqsubseteq} y \in C_s$ then we know that $x \equiv y$ hence $\alpha(x)$ satisfies the constraints $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} y$.

   Inductive Cases

   We prove the following cases by induction over the height of X.

(e) if $x \dot{\sqsubseteq} X,\ X \dot{\sqsubseteq} \exists f : \overline{Y} \in C_s$ then we know that:

i. either $xfy \in C_s$ such that $y \equiv \overline{Y}$ or $y \dot{\sqsubseteq} \overline{Y} \in C_s$

ii. or $x\ \exists f\ y \in C_s$ such that $y \equiv \overline{Y}$ or $y \dot{\sqsubseteq} \overline{Y} \in C_s$

hence if $\alpha(x)$ satisfies the constraint $y \dot{\sqsubseteq} \overline{Y}$ then $\alpha(x)$ satisfies the constraints $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \exists f : \overline{Y}$.

(f) if $x \dot{\sqsubseteq} X,\ X \dot{\sqsubseteq} \overline{Y} \sqcup \overline{Z} \in C_s$ then we can show that $\mathcal{R}, \alpha \models X \dot{\sqsubseteq} \overline{Y}$.

This case is analogous to case 2 above.

(g) if $x \dot{\sqsubseteq} X,\ X \dot{\sqsubseteq} \forall f : \overline{Y} \in C_s$ then we can show that $\mathcal{R}, \alpha \models X \dot{\sqsubseteq} \forall f : \overline{Y}$.

This case is analogous to case 3 above.

Hence, $\mathcal{R}, \alpha(x)$ satisfies all the constraints in C_s.

Hence, we have the theorem.

2.7.2 Normalisation Rules

This section presents a complete set of consistency preserving normalisation rules for any given system of constraints in the language $\mathcal{L}_2$ observing the acyclicity condition stated earlier. Our normalisation rules progressively transform any given system of $\mathcal{L}_2$ constraints into normal form and by the normal form theorem in theorem 2.7.8 the consistency of any system of constraints in normal form can be decided.

The process of transforming a given $\mathcal{L}_2$ constraint system into normal form is broken into 3 stages:

1. Simplifying negations
2. Breaking down complex $\dot{\sqsubseteq}$ constraints
3. Constraint propagation and simplification

The first stage is accomplished by the following transformation rules that simplify negated descriptions by effectively pushing negations inwards.

Stage 1: Push negations inwards

1. $\{\overline{X} \dot{\sqsubseteq} \neg\exists f : T\} \cup C_s \longrightarrow \{\overline{X} \dot{\sqsubseteq} \forall f : \neg T\} \cup C_s$
2. $\{\overline{X} \dot{\sqsubseteq} \neg\forall f : T\} \cup C_s \longrightarrow \{\overline{X} \dot{\sqsubseteq} \exists f : \neg T\} \cup C_s$
3. $\{\overline{X} \dot{\sqsubseteq} \neg(S \sqcap T)\} \cup C_s \longrightarrow \{\overline{X} \dot{\sqsubseteq} (\neg S \sqcup \neg T)\} \cup C_s$
4. $\{\overline{X} \dot{\sqsubseteq} \neg(S \sqcup T)\} \cup C_s \longrightarrow \{\overline{X} \dot{\sqsubseteq} (\neg S \sqcap \neg T)\} \cup C_s$
5. $\{\overline{X} \dot{\sqsubseteq} \neg(\neg S)\} \cup C_s \longrightarrow \{\overline{X} \dot{\sqsubseteq} S\} \cup C_s$
6. $\{\overline{X} \dot{\sqsubseteq} \neg\top\} \cup C_s \longrightarrow \{\overline{X} \dot{\sqsubseteq} \bot\} \cup C_s$
7. $\{\overline{X} \dot{\sqsubseteq} \neg\bot\} \cup C_s \longrightarrow \{\overline{X} \dot{\sqsubseteq} \top\} \cup C_s$

Once, the above rules have been completed the only negated descriptions that occur would be of the form $\neg x, \neg c$ or $\neg C$.

The next stage of the normalisation process attempts to simplify complex containment constraints into simpler ones by introducing new variables.

Stage 2 : Decompose $\mathcal{ALV}$ terms

1. $\{X \dot{\sqsubseteq} \overline{c}\} \cup C_s \longrightarrow \{X \dot{\sqsubseteq} x, x \dot{\sqsubseteq} \overline{c}\} \cup C_s$
   where x is new and $\overline{c}$ ranges over a, c
2. $\{x \dot{\sqsubseteq} \exists f : T\} \cup C_s \longrightarrow \{x\ \exists f\ y,\ y \dot{\sqsubseteq} T\} \cup C_s$
   where y is new
3. $\{X \dot{\sqsubseteq} \exists f : T\} \cup C_s \longrightarrow \{X \dot{\sqsubseteq} \exists f : Y,\ Y \dot{\sqsubseteq} T\} \cup C_s$
   where T is not a variable and Y is new
4. $\{\overline{X} \dot{\sqsubseteq} \forall f : T\} \cup C_s \longrightarrow \{\overline{X} \dot{\sqsubseteq} \forall f : Y,\ Y \dot{\sqsubseteq} T\} \cup C_s$
   where T is not a variable and Y is new
5. $\{\overline{X} \dot{\sqsubseteq} S \sqcap T\} \cup C_s \longrightarrow \{\overline{X} \dot{\sqsubseteq} S,\ \overline{X} \dot{\sqsubseteq} T\} \cup C_s$
6. $\{\overline{X} \dot{\sqsubseteq} S \sqcup T\} \cup C_s \longrightarrow \{\overline{X} \dot{\sqsubseteq} Y \sqcup Z, Y \dot{\sqsubseteq} S, Z \dot{\sqsubseteq} T\}$
   where S, T are not variables and Y and Z are new

7. $\{\overline{X} \dot{\sqsubseteq} y \sqcup T\} \cup C_s \longrightarrow \{\overline{X} \dot{\sqsubseteq} y \sqcup Z, Z \dot{\sqsubseteq} T\}$
   where T is not a variable and Z is new

8. $\{\overline{X} \dot{\sqsubseteq} S \sqcup z\} \cup C_s \longrightarrow \{\overline{X} \dot{\sqsubseteq} Y \sqcup z, Y \dot{\sqsubseteq} S\}$
   where S is not a variable and Y is new

Once the above rules have run to completion every containment constraint involving big variables would be of the form $X \dot{\sqsubseteq} \forall f : \overline{Y}, X \dot{\sqsubseteq} \exists f : \overline{Y}, X \dot{\sqsubseteq} x, X \dot{\sqsubseteq} \neg x, X \dot{\sqsubseteq} C, X \dot{\sqsubseteq} \neg C, X \dot{\sqsubseteq} \neg c, X \dot{\sqsubseteq} \neg a$.

Note that constraints of the form $X \dot{\sqsubseteq} c$ and $X \dot{\sqsubseteq} a$ are eliminated by introducing new variables.

The rules in Stage 3 do the actual constraint propagation on a set of constraints that have been simplified by the rules in the previous two stages. In order to guarantee the termination of the Stage 3 rules, it turns out to be necessary to impose fairly strict ordering restrictions. For this purpose the set of rules is broken down into 3 groups.

1. *Group 0* rules break down containment constraints of the form $x \dot{\sqsubseteq} \Gamma$ whenever possible.

2. *Group 1* rules are essentially the $\mathcal{L}_1$ normalisation rules that we discussed in section 2.4.

3. *Group 2* rules push X-constraints through a variable x in the presence of the constraints of the form $x \dot{\sqsubseteq} X$.

We assume that the Stage 3 rules are applied with the following order of priority.

1. **Group 0** rules are applied whenever possible and hence with the highest priority

2. **Group 1** rules are applied next.

3. **Group 2** rules are ordered. This means that rule **BDis** is applied first (if applicable) whereas rule **BExists** is applied last.

Now we are ready to present the constraint propagation rules for the language $\mathcal{L}_2$.

Stage 3 : Constraint Propagation rules

Group 0 : Rules to simplify $\dot{\sqsubseteq}$ constraints

(ExistsE) $\{x \dot{\sqsubseteq} \exists f : \overline{Y}\} \cup C_s \longrightarrow \{x\ \exists f\ y,\ y \dot{\sqsubseteq} \overline{Y}\} \cup C_s$
where y is new

(ForallE) $\{x \dot{\sqsubseteq} \forall f : y\} \cup C_s \longrightarrow \{x\ \forall f\ y\} \cup C_s$

(EqualsI) $\{x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} y\} \cup C_s \longrightarrow [x/X]\{x = y\} \cup C_s$

Group 1 : Rules to simplify $\mathcal{L}_1$ constraints

(Equals) $\{x = y\} \cup C_s \longrightarrow \{x = y\} \cup [x/y]C_s$
if $x \not\equiv y$ and y occurs in C_s

(Const) $\{x \dot{\sqsubseteq} \overline{c}, y \dot{\sqsubseteq} \overline{c}\} \cup C_s \longrightarrow \{x = y, x \dot{\sqsubseteq} \overline{c}\} \cup C_s$
where $\overline{c}$ ranges over a, c

(Feat) $\{xfy,\ xFz\} \cup C_s \longrightarrow \{xfy,\ y = z\} \cup C_s$
where F ranges over $f, \exists f, \forall f$

(ExForall) $\{x\ \exists f\ y,\ x\ \forall f\ z\} \cup C_s \longrightarrow \{xfy,\ y = z\} \cup C_s$

(ExistsF) $\{x\ \exists f\ y\} \cup C_s \longrightarrow \{xfy\} \cup C_s$
if $f \in \mathcal{F}s$

(EqualsE) $\{x \dot{\sqsubseteq} y\} \cup C_s \longrightarrow \{x = y\} \cup C_s$

(Neg) $\{x \dot{\sqsubseteq} \neg y\} \cup C_s \longrightarrow \{x \neq y\} \cup C_s$

Group 2: Rules for propagating $\dot{\sqsubseteq}$ constraints

(BDis) $\{\overline{X} \dot{\sqsubseteq} \overline{Y} \sqcup \overline{Z}\} \cup C_s \longrightarrow \{\overline{X} \dot{\sqsubseteq} \overline{Y} \sqcup \overline{Z}, x \dot{\sqsubseteq} \overline{W}\} \cup C_s$
if :

1. $x \dot{\sqsubseteq} \overline{Y} \notin C_s$, $x \dot{\sqsubseteq} \overline{Z} \notin C_s$, $x \not\equiv \overline{Y}$, $x \not\equiv \overline{Z}$
2. $\overline{W} \equiv \overline{Y}$ or $\overline{W} \equiv \overline{Z}$
3. and either of

(a) $\overline{X} \equiv x$ *or*

(b) $\overline{X} \equiv X$ *and* $x \dot{\sqsubseteq} X \in C_s$

Note that this rule is *non-deterministic.*

(BConst) $\{x \dot{\sqsubseteq} X,\ X \dot{\sqsubseteq} \overline{C}\} \cup C_s \longrightarrow \{x \dot{\sqsubseteq} X,\ x \dot{\sqsubseteq} \overline{C},\ X \dot{\sqsubseteq} \overline{C}\} \cup C_s$
if $x \dot{\sqsubseteq} \overline{C} \notin C_s$ where $\overline{C}$ ranges over $\neg x, C, \neg C$ and $\neg c$

(BForall) $\{\overline{X} \dot{\sqsubseteq} \forall f : \overline{Y},\ x\ F\ y\} \cup C_s \longrightarrow \{\overline{X} \dot{\sqsubseteq} \forall f : \overline{Y},\ x\ F\ y, y \dot{\sqsubseteq} \overline{Y}\} \cup C_s$
if F ranges over f and $\exists f,\ y \dot{\sqsubseteq} \overline{Y} \notin C_s,\ y \not\equiv \overline{Y}$ and either of:

1. $\overline{X} \equiv x$ *or*
2. $\overline{X} \equiv X$ and $x \dot{\sqsubseteq} X \in C_s$

(BExists) $\{x \dot{\sqsubseteq} X,\ X \dot{\sqsubseteq} \exists f : \overline{Y}\} \cup C_s \longrightarrow \{x \dot{\sqsubseteq} X,\ X \dot{\sqsubseteq} \exists f : \overline{Y},\ x \exists f y,\ y \dot{\sqsubseteq} \overline{Y}\} \cup C_s$
if y is new and there is no z such that $xFz \in C_s$ where F ranges over $f, \exists f$ and either of

1. $z \dot{\sqsubseteq} \overline{Y} \in C_s$ *or*
2. $z \equiv \overline{Y}$

All the rules in Group 1 are the same as the normalisation rules for the language $\mathcal{L}_1$ except for rule **ExistsF** which is needed since $\mathcal{ALV}$ permits relations $f \in \mathcal{F}s$ to be treated as functional.

Rules in Group 2 are mainly intended to push constraints attached to a big variable X through a variable x in the presence of constraints of the form $x \dot{\sqsubseteq} X$. These again transform a given constraint system into normal form as required by definition 2.7.7. It is important to note that while all rules in Group 2 generates new constraints, it is only rules **ExistsE** and **BExists** (among all the rules in Group 0 through Group 2) that generate new variables. This fact turns out to be important for choice of our rule ordering and hence in proving termination.

Our rules and their ordering follow the following philosophy:

1. Eliminate containment constraints as eagerly as possible - rules **ExistsE** and **ForallE**.

2. Keep the number of new variables generated minimal - rule **BExists** which is applied last and hence delayed as much as possible.

3. Keep the number of big variables to a minimum by eliminating big variables as eagerly as possible - rule **EqualsI**.

4. Keep all the $\mathcal{L}_1$ constraints in normal form - all the rules in Group 1.

To illustrate the way our rules transform a given $\mathcal{L}_2$ constraint system into normal form we shall look at an example.

Example : Let the initial $\mathcal{L}_2$ constraint system be :

$$\{x_0 \dot{\sqsubseteq} (x \sqcap \exists f : \top \sqcap \forall f : ((x \sqcup u) \sqcap \exists f : \top))\}$$

After applying the simplification rules of Stage 2, we get:

$$\underline{x_0 \dot{\sqsubseteq} x}, x_0 \exists f\ y, y \dot{\sqsubseteq} \top, x_0 \dot{\sqsubseteq} \forall f : Y, Y \dot{\sqsubseteq} x \sqcup u, Y \dot{\sqsubseteq} \exists f : Z, Z \dot{\sqsubseteq} \top$$

After eliminating x_0 by applying rule **EqualsE** and rule **Equals** we get:

$$x \exists f\ y, \underline{y \dot{\sqsubseteq} \top}, x \dot{\sqsubseteq} \forall f : Y, Y \dot{\sqsubseteq} x \sqcup u, Y \dot{\sqsubseteq} \exists f : Z, \underline{Z \dot{\sqsubseteq} \top}$$

Since the constraints $y \dot{\sqsubseteq} \top$ and $Z \dot{\sqsubseteq} \top$ will not play any major part, we shall remove them from consideration. Let C_{s0} be the new constraint system.

Applying the normalisation rules in Stage 3 we get the following simplified constraint systems:

1. $C_{s0} = \{\underline{x\ \exists f\ y}, \underline{x \dot{\sqsubseteq} \forall f : Y}, Y \dot{\sqsubseteq} x \sqcup u, Y \dot{\sqsubseteq} \exists f : Z\}$

2. Add $y \dot{\sqsubseteq} Y$ (rule **BForall**) to get:
   $C_{s1} = \{x\ \exists f\ y, x \dot{\sqsubseteq} \forall f : Y, \underline{Y \dot{\sqsubseteq} x \sqcup u}, Y \dot{\sqsubseteq} \exists f : Z, \underline{y \dot{\sqsubseteq} Y}\}$

3. Add $y = x$ (rule **BDis**)

4. Rewriting y to x (rule **Equals**) we get:
   $C_{s3} = \{x\ \exists f\ x, x \dot{\sqsubseteq} \forall f : Y, Y \dot{\sqsubseteq} x \sqcup u, \underline{Y \dot{\sqsubseteq} \exists f : Z}, \underline{x \dot{\sqsubseteq} Y}\}$

5. Add $x\ \exists f\ z, z \dot{\sqsubseteq} Z$ (rule **BExists**) to get:
   $C_{s4} = \{x\ \exists f\ x, \underline{x \dot{\sqsubseteq} \forall f : Y}, Y \dot{\sqsubseteq} x \sqcup u, Y \dot{\sqsubseteq} \exists f : Z, x \dot{\sqsubseteq} Y, \underline{x\ \exists f\ z}, z \dot{\sqsubseteq} Z\}$

6. Add : $z \dot{\sqsubseteq} Y$ (rule **BForall**)

7. Add : $z = x$ (rule **BDis**)

8. Rewriting z to x (rule **Equals**) we get :
   $C_{s5} = \{x\ \exists f\ x, x \dot{\sqsubseteq} \forall f : Y, Y \dot{\sqsubseteq} x \sqcup u, Y \dot{\sqsubseteq} \exists f : Z, x \dot{\sqsubseteq} Y, x\ \exists f\ x, x \dot{\sqsubseteq} Z\}$

It is easy to verify that the final resulting constraint system is in normal form and no further rules apply. Of course, there is one more completion generated by rule **BDis** on the constraint $Y \dot{\sqsubseteq} x \sqcup u$ that we have not considered.

On the other hand, to see that the ordering we have chosen is crucial to guarantee termination consider the following simplification steps in which we execute rule **BExists** while some other Group 1/Group 2 rules are still applicable. We show that in this case non-termination results:

1. $C_{s0} = \{\underline{x\ \exists f\ y}, \underline{x \dot{\sqsubseteq} \forall f : Y}, Y \dot{\sqsubseteq} x \sqcup u, Y \dot{\sqsubseteq} \exists f : Z\}$

2. Add $y \dot{\sqsubseteq} Y$ (rule **BForall**) to get:
   $C_{s1} = \{x\ \exists f\ y, x \dot{\sqsubseteq} \forall f : Y, Y \dot{\sqsubseteq} x \sqcup u, \underline{Y \dot{\sqsubseteq} \exists f : Z}, \underline{y \dot{\sqsubseteq} Y}\}$

3. Now instead of applying rule **BDis** to add $y \dot{\sqsubseteq} x$ we apply rule **BExists** to get:
   $C_{s2} = \{x\ \exists f\ y, x \dot{\sqsubseteq} \forall f : Y, \underline{Y \dot{\sqsubseteq} x \sqcup u}, Y \dot{\sqsubseteq} \exists f : Z, \underline{y \dot{\sqsubseteq} Y}, y\ \exists f\ z,\ z \dot{\sqsubseteq} Z\}$

4. Add $y \dot{\sqsubseteq} x$ (rule **BDis**)

5. Rewriting y to x (rules **EqualsE** and **Equals**) we get:
   $C_{s3} = \{x\ \exists f\ x, \underline{x \dot{\sqsubseteq} \forall f : Y}, Y \dot{\sqsubseteq} x \sqcup u, Y \dot{\sqsubseteq} \exists f : Z, x \dot{\sqsubseteq} Y, \underline{x\ \exists f\ z},\ z \dot{\sqsubseteq} Z\}$

6. Add $z \dot{\sqsubseteq} Y$ (rule **BForall**) to get:
   $C_{s4} = \{x\ \exists f\ x, x \dot{\sqsubseteq} \forall f : Y, Y \dot{\sqsubseteq} x \sqcup u, \underline{Y \dot{\sqsubseteq} \exists f : Z}, x \dot{\sqsubseteq} Y, x\ \exists f\ z,\ z \dot{\sqsubseteq} Z,\ \underline{z \dot{\sqsubseteq} Y}\}$

7. Now instead of applying rule **BDis** (which is applicable in the current state) to add $z \dot{\sqsubseteq} x$ we apply rule **BExists**. This adds $z \exists f\ z_1, z_1 \dot{\sqsubseteq} Z$ to get:

$C_{s4} = \{x\ \exists f\ x, x \dot{\sqsubseteq} \forall f : Y, \underline{Y \dot{\sqsubseteq} x \sqcup u}, Y \dot{\sqsubseteq} \exists f : Z, x \dot{\sqsubseteq} Y, x\ \exists f\ z,\ z \dot{\sqsubseteq} Z,\ \underline{z \dot{\sqsubseteq} Y}, z \exists f\ z_1, z_1 \dot{\sqsubseteq} Z\}$

8. Add $z \dot{\sqsubseteq} x$ (rule **BDis**)

9. Rewriting z to x (rules **EqualsE** and **Equals**) we get:
$C_{s6} = \{\ x\ \exists f\ y, \underline{x \dot{\sqsubseteq} \forall f : Y}, Y \dot{\sqsubseteq} x \sqcup u, Y \dot{\sqsubseteq} \exists f : Z, x \dot{\sqsubseteq} Y,$
$x\ \exists f\ x,\ x \dot{\sqsubseteq} Z,\ x \dot{\sqsubseteq} Y, \underline{x \exists f\ z_1}, z_1 \dot{\sqsubseteq} Z\}$

10. The situation now is analogous to C_{s3} and the process can go forever.

2.8 Invariance, Completeness and Termination

The notion of invariance we employ for $\mathcal{L}_2$ is a weaker one as compared to the notion we adopted for $\mathcal{L}_1$. This is due to two reasons. Firstly, $\mathcal{L}_2$ provides disjunctions and hence not every instance of the normalisation rule is consistency preserving. Secondly, $\mathcal{L}_2$ normalisation rules introduce new variables and hence only the interpretation of the initial set of variables is preserved.

Theorem 2.8.1 [Invariance] *C_s is a consistent constraint system iff at least one instance of every applicable rule transforms C_s into a consistent constraint system.*

Proof: Although it is quite easy to see that all the rules presented in Stage 1, 2 and 3 are invariant we shall dwell on some important ones.

One question that arises regarding Stage 2 rules 6 through 8 as to why these cannot be replaced by a single rule of the form:

(23) $\{x \dot{\sqsubseteq} S \sqcup T\} \cup C_s \longrightarrow \{x \dot{\sqsubseteq} y \sqcup z, y \dot{\sqsubseteq} S, z \dot{\sqsubseteq} T\} \cup C_s$

We shall show that rule 23 is not invariant given our semantics. To show that this is indeed the case it is enough to consider the following example constraint system which is clearly consistent:

$C_s = \{x \dot{\sqsubseteq} (x \sqcap a) \sqcup (x \sqcap b)\}$

Applying the rule given in 23 we get:

- $\{x \dot{\sqsubseteq} y \sqcup z, y \dot{\sqsubseteq} (x \sqcap a), z \dot{\sqsubseteq} (x \sqcap b)\}$

This results in the following simplification steps:

- $\{x \dot{\sqsubseteq} y \sqcup z, y \dot{\sqsubseteq} x, y \dot{\sqsubseteq} a, z \dot{\sqsubseteq} x, z \dot{\sqsubseteq} b\}$
- $\{x \dot{\sqsubseteq} x \sqcup x, x \dot{\sqsubseteq} a, x \dot{\sqsubseteq} b\}$
  which is inconsistent

Hence, one of our normalisation steps is incorrect. In fact, our new rule 23 is not invariant since it assumes that both S and T are consistent *i.e.* their denotations are non-empty but this is incorrect since for $x \dot{\sqsubseteq} S \sqcup T$ to be consistent it is enough that either S or T is consistent.

Next, the presence of disjunctions in a $\mathcal{L}_2$ constraint system means that not every application of rule **BDis** results in a consistency preserving translation of the original constraint system. Hence, the invariance claim is a weaker claim in that our normalisation procedure searches through the disjunctions until a consistent completion is found or all the choice points are exhausted.

To prove the invariance claim it is enough to note that all rules except rule **BDis** preserves both inconsistency and consistency. Rule **BDis** on the other hand preserves consistency in at least one of the two choices chosen while preserving inconsistency in each of the choices.

Theorem 2.8.2 [Completeness of Rewriting] *To every constraint system not in normal form one of the normalisation rules applies.*

Proof: To prove the completeness claim, as for the language $\mathcal{L}_1$, we need to ensure that if a constraint system is not in normal form then one of the normalisation rules apply. For each of the conditions for normal form given in definition 2.3.2, it is easy to see that there is a corresponding rule.

Hence we have the theorem.

2.8.1 Termination

Next, we establish an important result of this chapter namely that our normalisation procedure terminates and hence testing the consistency of $\mathcal{ALV}$ terms is decidable. However since the termination proof is tedious and long we have delegated it to Appendix B.

Intuitively speaking, it is clear that both Stage 1 and Stage 2 rules terminate in linear time. For Stage 3 rules, if we omit rule **BExists** then there would be no rules that generate new variables. Since there are a fixed number of big variables and a fixed number of constraints of the form $X \dot{\sqsubseteq} \Gamma$ it is intuitively clear that this process must also terminate.

The difficulty arises when we take into account rule **BExists** since it adds new variables. We show in Appendix B that taking this factor into account our normalisation rules terminates.

Theorem 2.8.3 [Termination] *There is no infinite chain of rule applications issuing from any completion of the normalisation rules on any finite $\mathcal{L}_2$ constraint system.*

2.8.2 Summary of Results

As a corollary to the termination result we have:

Corollary 2.8.4 *There is no infinite chain issuing from the application of the normalisation rules on any initial constraint system of the form $\{x \dot{\sqsubseteq} T\}$.*

This follows since the break down of a constraint of the form $x \dot{\sqsubseteq} T$ involving an $\mathcal{ALV}$ term T always involves big variables obeying the acyclicity condition.

Since there is only a finite number of ways in which each normalisation rule can be instantiated we have the following corollary.

Corollary 2.8.5 *The consistency of any $\mathcal{ALV}$ term is decidable.*

Lemma 2.8.6 *Deciding consistency of $\mathcal{ALV}$ terms is PSPACE-hard.*

Proof: We know that deciding consistency of $\mathcal{ALC}$ terms is a PSPACE complete problem

[Schmidt-Schauß & Smolka 91]. Since every $\mathcal{ALC}$ term is also an $\mathcal{ALV}$ term, it follows that deciding consistency of $\mathcal{ALV}$ terms is PSPACE-hard.

However, a more complete picture of the computational complexity of consistency testing in $\mathcal{ALV}$ goes beyond the scope of the current thesis and is left as a future exercise.

2.9 An undecidability result

In this section we shall show that if we remove the restriction imposed by the *acyclicity condition* on the constraint language $\mathcal{L}_2$ and add the provision of path equations where the paths are restricted to lists of feature symbols then consistency testing of $\mathcal{L}_2$ constraint systems becomes *undecidable*. We prove this result by translating the *word-problem* for semi-groups which in known to be undecidable [Gurevich 66] (see also [Boone 59], [Novikov 55]) into a $\mathcal{L}_2$ constraint system. Our proof is based on related undecidability results due to [Dörre & Rounds 90] and [Schmidt-Schauß 89]. In [Dörre & Rounds 90], it is shown that the addition of *subsumption constraints* to feature clauses renders consistency testing undecidable. However, it should be noted that their usage of constraints of the form $x \dot{\sqsubseteq} y$ (which they call subsumption constraints) is essentially different from our containment constraints[1]. In [Schmidt-Schauß 89], it is shown that the addition of the so called *role-value maps* (which can be thought of as the relational counterpart of path equations where each path is a list of *relation* symbols) renders determination of subsumption in a small subset of KL-ONE (containing *no* unions or negations) undecidable.

First of all a **path equation** is a term of the form :

$$P = Q$$

where both P and Q are lists of *feature* symbols.

Assume that the above construct is available in the term language $\mathcal{ALV}$.

[1]This can be trivially demonstrated by the fact that containment constraints such as $x \dot{\sqsubseteq} y$ is equivalent to $x = y$ whereas this is not the case with subsumption constraints.

The semantics of $\mathcal{ALV}$ path equations is given by the following definition with respect to an relational algebra $\mathcal{I}$:

$$[\![P = Q]\!]^{\mathcal{I}} = \{e \in \mathcal{U}^I \mid P^I(e) = Q^I(e)\}$$

where lists of feature symbols are interpreted as function (or relational) composition of the component feature symbols.

Now, we are ready to prove the undecidability result.

The *word-problem* for finite semi-groups can be stated as follows.

Let Σ be an alphabet consisting of atomic symbols. Let, Σ^* be the set of finite strings drawn from Σ. By overloading the symbol Σ^*, let Σ^* be the semigroup under string concatenation. Let E be a finite set of equations of the form $P_i = Q_i$ such that $P_i, Q_i \in \Sigma^+$. Let $P = Q$ be another such equation. Then the word problem is to decide whether every finite semigroup that satisfies every equation in E satisfies $P = Q$.

We now show that this problem can be encoded in a $\mathcal{L}_2$ constraint system as follows.

Let $\Sigma \subseteq \mathcal{F}s$ *i.e.* treat every symbol in Σ as a feature symbol.

Construct a constraint system C_s as a union of the following constraint systems:

1. $C_{s1} = \{x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \exists f : X \mid f \in \Sigma\}$
2. $C_{s2} = \{x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} P_i = Q_i \mid P_i = Q_i \in E\}$
3. $C_{s3} = \{x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \neg P = Q, \}$

i.e. $C_s = C_{s1} \cup C_{s2} \cup C_{s3}$

Note the crucial violation of the acyclicity condition in the definition of C_{s1}.

Lemma 2.9.1 *For any relational algebra $\mathcal{I}$ and a variable assignment α satisfying C_{s1} it follows that for any $x \in \alpha(X) : P^I(x) \downarrow$ for every $P \in \Sigma^*$.*

Proof: By induction on the length of P. Note that for every $X \dot{\sqsubseteq} \exists f : X \in C_{s1} : f$ is functional.

Lemma 2.9.2 *For any relational algebra $\mathcal{I}$ and a variable assignment α satisfying C_{s1}, for every $x \in \alpha(X)$: if $U^I(x) = V^I(x)$ then $(UP)^I(x) = (VP)^I(x)$*

Proof: By induction on the length of P.

Theorem 2.9.3 *The constraint system C_s is satisfiable iff there exists a finite semigroup satisfying every equation in E but not the equation $P = Q$.*

Proof: For the first part if every finite semigroup satisfying every equation in E also satisfies $P = Q$ then we shall show that there is no finite relational algebra and an assignment that satisfies C_s. Assume that there exists a *finite* relational algebra[2] $\mathcal{I}$ and a $\mathcal{I}$-assignment α which satisfy C_{s1} and C_{s2}. Construct a finite semigroup $< S^I, . >$ as follows. Let J^I be the set defined by:

$$J^I = \{x \in \mathcal{U}^I \mid \exists W \in \Sigma^+ \text{ such that } W^I(x) \downarrow\}$$

Let $[U]$ be the equivalence class of paths given by the following definition :

$$[U] = \{V \in \Sigma^+ \mid \ \forall x \in J^I : U^I(x) = V^I(x)\}$$

Let S^I be the set of equivalence classes of paths given by :

$S^I = \{[U] \mid U \in \Sigma^+\} \cup \{[Id_I]\}$
where Id_I is the *identity* function $Id_I : J^I \longrightarrow J^I$

By overloading, let . be the concatenation operation defined on elements on S^I by $[U].[V] = [UV]$.

We claim that the structure $< S^I, . >$ is a semigroup as follows :

1. $[P] \in S^I$ for every $P \in \Sigma^*$ (*Lemma* 2.9.1)
2. if $[U] = [V]$ then for any $P \in \Sigma^*$:
   (a) $[UP] = [VP]$
   Proof: By Lemma 2.9.2.

[2]We assume that a relational algebra $\mathcal{I}$ is finite just in case its universe $\mathcal{U}^I$ is finite.

(b) $[PU] = [PV]$

Proof: For any $x \in \alpha(X) : P^I(x) \downarrow$.

Let $P^I(x) = y$, then since $y \in \alpha(X)$, we have $U^I(x) \downarrow$, $V^I(x) \downarrow$ by Lemma 2.9.1.

By hypothesis, $U^I(y) = V^I(y)$. This means $(PU)^I(x) = (PV)^I(x)$.

Since, the above is valid for every $x \in \alpha(X)$, we get $[PU] = [PV]$.

This proves that $< S^I, . >$ is a semigroup.

We now show that $< S^I, . >$ satisfies every equation in E as follows :

for every $P_i = Q_i$ in E we know that every element $x \in \alpha(X)$ satisfies $P_i^I(x) = Q_i^I(x)$.

Hence, $[P_i] = [Q_i]$.

By hypothesis $< S^I, . >$ satisfies the equation $P = Q$. In order words, for every element $x \in \alpha(X)$ we have $P^I(x) = Q^I(x)$.

Hence, there exists no interpretation $\mathcal{I}, \alpha$ satisfying C_s.

For the second part, assume that there exists a finite semigroup $< G, . >$ (where the elements of G are the equivalence classes of elements in Σ^*) satisfying every equation in E but *not* $P = Q$, then we need to show that there exists an interpretation $\mathcal{J}, \beta$ which satisfies C_s.

We construct $\mathcal{J}, \beta$ as follows :

1. Let $\mathcal{U}^J = \{[P] \mid P \in G\}$
   i.e. the set of distinct equivalence classes in G
2. Interpret every feature f by :
   $f^J([P]) := [f.P]$
3. Let $\beta(x) := [Id_G]$ *i.e.* interpret x by the semigroup identity.
4. Let $\beta(X) := \mathcal{U}^J$ *i.e.* interpret X by the distinct equivalence classes in G.

From the above construction $\mathcal{J}, \beta$ satisfies every equation in E since:

1. for every $e \in \beta(X) : f^J(e) \downarrow$ for every $f \in \Sigma$.

   Hence $\mathcal{J}, \beta$ satisfies C_{s1}.

2. for every $P_i = Q_i$ in E :

   (a) $[P_i] = [Q_i]$

   (b) $[P_i.U] = [Q_i.U]$ since $\mathcal{J}$ inherits the semigroup property of G.

   Hence, $\mathcal{J}, \beta$ satisfies C_{s2}.

3. By hypothesis $[P] \neq [Q]$, hence $\mathcal{J}, \beta$ satisfies C_{s3}.

This proves that $\mathcal{J}, \beta$ satisfies C_s.

Hence, we have the theorem.

Corollary 2.9.4 *The addition of path equations and the removal of the acyclicity condition to the language $\mathcal{L}_2$ renders consistency testing undecidable even if paths are restricted to feature symbols.*

2.9.1 Summary and Related Work

In this section we have shown that the addition of cyclic coreferences of big variables and path constraints renders consistency checking of $\mathcal{L}_2$ constraints undecidable. This holds even if paths are restricted to lists of feature symbols. However, it leaves open the question as to whether leaving only one of the two conditions makes consistency testing decidable.

Assume that we retain the *acyclicity condition* but allow path equations in $\mathcal{L}_2$ where paths are restricted to functional chains. From the constraint propagation rules for the language $\mathcal{L}_2$, it can be seen that path equations introduce a finite number of new variables such that each of the new variables cannot produce the same path equation and furthermore these new variables cannot interact with any constraint of the form $X \doteq \exists f : Y$ or $X \doteq \forall f : Y$ to produce additional new variables since each of the new variables are part of a functional path. With these considerations we conjecture that

addition of path equations over functional chains alone to $\mathcal{ALV}$ is insufficient to cause undecidability.

The notion of cyclic coreferences of big variables is closely related to the notion of terminological cycles (see [Nebel 90], [Nebel 91], [Baader 90]) contained in *sort definitions*. In [Baader *et al* 91] it is shown that consistency testing of feature terms augmented with *sort equations* (which generalises sort definitions) can be reduced to inconsistency testing of terms containing so called *functional uncertainty* [Kaplan & Bresnan 82], [Kaplan & Maxwell III 89]. However, surprisingly the addition of *regular expressions* over role symbols (which can be thought as the relational counterpart of functional uncertainty) to the language $\mathcal{ALC}$ [Schmidt-Schauß & Smolka 91] does not render consistency testing undecidable as shown by *Baader* [Baader 91] which provides a consistency testing algorithm for the augmented language. *Baader* also shows that one can translate *cyclic* $\mathcal{ALC}$ terminologies to *acyclic* terminologies containing regular path expressions as long as cyclic terminologies are interpreted using the *descriptive semantics* of [Nebel 91]. From this result we conjecture that the addition of cyclic coreferences of big variables (while disallowing path equations) in $\mathcal{L}_2$ would still make consistency testing *decidable*.

The above relationship can also be seen in the light of work by *Schild* [Schild 91] who shows that a large number of concept languages are notational variants of *polymodal logics* and *propositional dynamic logics* [3]. This correspondence has shown that the addition of *role conjunctions* and feature symbols to a (propositionally complete) concept language that permits regular expressions of role symbols renders consistency testing *undecidable* while surprisingly prohibiting feature symbols in the same language makes consistency testing *decidable*.

[3] However note that there is no direct one-to-one correspondence between $\mathcal{ALV}$ on one hand and modal and propositional dynamic logics investigated by [Schild 91] on the other since the latter lacks variables. This distinguishes $\mathcal{ALV}$ from these logics.

2.10 Summary and Discussion

In this Chapter we have shown that the inclusion of unquantified variables in a concept language containing unions, complements, negations of relations and feature descriptions does not render the logic undecidable. Our logic $\mathcal{ALV}$ is aimed primarily for computational linguistic applications although it is quite appropriate for general knowledge representation tasks.

Firstly, we have shown that the notion of *feature algebras* [Smolka 89] generalises quite naturally to our *relational algebras* and the notion of *partial homomorphisms* extends conservatively to relational algebras. Secondly, the notion of *feature graph algebras* extends naturally to *relational graph algebras* which provide canonical interpretations to the language $\mathcal{L}_1$ which again extends Smolka's feature clauses.

Our consistency testing algorithms for $\mathcal{ALV}$ show that due to the presence of variables great care has to be taken to avoid non-termination. This is self evident in the strict rule ordering our termination proof imposes. However, in terms of a real implementation this simply means that variable equalities have to be generated and eliminated as early as possible while generation of new variables has to be delayed as much as possible. This we believe will not impose any additional implementational difficulties since most constraint solvers are built around the idea of keeping the constraint system as simple as possible by eliminating variable equalities.

For knowledge representation systems that use *classification* as the sole computational service, the logic $\mathcal{ALV}$ may give some unusual results. This is due to the fact that in $\mathcal{ALV}$ the term $x \sqcap \neg y$ (where x and y are distinct variables) is consistent. This means that testing subsumption between terms may not give the desired results. To illustrate this more concretely, let us look at an example. Say, we want to test whether $(\exists \mathit{likes} : x \sqcap \exists \mathit{his\_dog} : x)$ *subsumes* $(\exists \mathit{likes} : y \sqcap \exists \mathit{his\_dog} : y)$. We know that $S \sqsubseteq T$ is true *iff* the term $\neg S \sqcap T$ is *inconsistent.* Hence, $(\exists \mathit{likes} : x \sqcap \exists \mathit{his\_dog} : x)$ *subsumes* $(\exists \mathit{likes} : y \sqcap \exists \mathit{his\_dog} : y)$ iff $\neg(\exists \mathit{likes} : x \sqcap \exists \mathit{his\_dog} : x) \sqcap (\exists \mathit{likes} : y \sqcap \exists \mathit{his\_dog} : y)$ is inconsistent. This is equivalent to $(\forall \mathit{likes} : \neg x \sqcup \forall \mathit{his\_dog} : \neg x) \sqcap (\exists \mathit{likes} : y \sqcap \exists \mathit{his\_dog} : y)$ which is consistent. This means that computing classification which essentially

involves testing for subsumption may not give the desired results in $\mathcal{ALV}$. Since most natural language applications do not require classification the above problem does not arise for these tasks.

However, since $\mathcal{ALV}$ provides constants and atoms, classification of terms involving constants and atoms but not variables will provide the expected results.

We believe that the work described in this Chapter represents the first major step towards achieving the ultimate goal of developing a constraint logic both for computational linguistics and knowledge representation tasks.

Chapter 3

A Concept Language with Partial Set Descriptions

3.1 Introduction

In this Chapter our aim is to extend the logic $\mathcal{ALV}$ that we developed in the last Chapter to include the so called *partial descriptions of sets* or *set descriptions* [Pollard & Sag 87],[Pollard & Moshier 90]. Set descriptions have important applications in computational linguistics such as in the grammatical framework HPSG [Pollard & Sag 87] [Pollard & Sag 92] but their computational characterisation largely remains unformalised. Although set-values have been studied by [Rounds 88] and [Pollard & Moshier 90] their work does not provide a computational characterisation of set-values. In particular, no constraint logic has been developed for dealing with set-values. Similarly, there has not been any concrete study relating set-values with concept languages, which we show have a surprisingly close connection with them. Furthermore, while [Rounds 88] provides models for set-values in terms of finite state automata (FSA) and defines a satisfaction relation between a FSA and KR-logic ([Kasper & Rounds 86]) formulas extended with set-values, it does not provide any consistency testing algorithms in this extended logic. On the other hand, [Pollard & Moshier 90] studies the domain theoretic properties of HPSG set descriptions in terms of powerdomains (see [Main 87] for an introduction) while making no attempt at either extending KR-logic or providing a consistency testing algorithm in

some extended logic.

Since our principal motivation is to provide advanced knowledge representation tools for Natural Language processing, we shall focus on set values both from a logical and computational perspective.

Our strategy for incorporating set descriptions (as we shall call them) is described in the following which will also provide a broad outline of this Chapter.

Firstly, we shall augment the constraint language $\mathcal{L}_1$ with an additional type of constraint to provide a datastructure representation for sets. Secondly, we shall extend the $\mathcal{L}_1$ constraint solving machinery to provide consistency testing of $\mathcal{L}_1$ constraints containing constraints describing sets. For this purpose, we need to add additional types of constraints.

Secondly, we shall extend the term description language $\mathcal{ALV}$ into the language $\mathcal{ALS}$ (an *acronym* for *Attributive Logic with Set Descriptions*) which provide set descriptions and negated set descriptions - something that has not been considered previously within computational linguistics. We shall show that the logic $\mathcal{ALS}$ has a close analog with concept languages containing so called *number restrictions* [Hollunder & Nutt 90]. However, since set descriptions are a natural knowledge representation structure, we shall argue that the provision of set descriptions provides a compact and more efficient representation as opposed to concept languages providing number restrictions. Importantly, since $\mathcal{ALS}$ provides variables in addition to set descriptions it is possible to state variable co-references within set descriptions - something that decidable concept languages have lacked so far but which is an important requirement for NL applications. To enable consistency testing of terms in $\mathcal{ALS}$ we shall augment the language $\mathcal{L}_2$ appropriately to provide consistency testing of $\mathcal{ALS}$ terms and show that our constraint solving process is invariant, complete and terminating. This proves that $\mathcal{ALS}$ is a *decidable* logic.

3.2 Motivation

In this section we look at some representative natural language applications for set values. The potential application for set values is much broader than that portrayed in the following examples. Some of these we deal with in more detail when we look at linguistic applications in Chapter 6.

Our first example from HPSG ([Pollard & Sag 87] *pp.* 104) restated from Chapter 1 shows the usage of set-values for modelling so called *semantic indices.* The following attribute-value matrix notation is intended to represent the semantics of the sentence *Kim sees Sandy.*

$$\left[\begin{array}{ll} \text{CONT} & \left[\begin{array}{ll} \text{REL} & \textit{see} \\ \text{SEER} & \boxed{2} \\ \text{SEEN} & \boxed{1} \end{array}\right] \\ \text{INDS} & \left\{\left[\begin{array}{ll} \text{VAR} & \boxed{1} \\ \text{REST} & \left[\begin{array}{ll} \text{RELN} & \textit{naming} \\ \text{NAME} & \textit{sandy} \\ \text{NAMED} & \boxed{1} \end{array}\right] \end{array}\right], \left[\begin{array}{ll} \text{VAR} & \boxed{2} \\ \text{REST} & \left[\begin{array}{ll} \text{RELN} & \textit{naming} \\ \text{NAME} & \textit{kim} \\ \text{NAMED} & \boxed{2} \end{array}\right] \end{array}\right]\right\} \end{array}\right]$$

The value of the CONT label represents the semantics of the sentence which roughly means $see(kim, sandy)$. The value of the INDS label is a set known as the *semantic indices* which denotes the objects involved in the *seeing* situation namely *Kim* and *Sandy.* The boxed numbers [1] and [2] represent variables and serve as co-reference markers. Semantic indices in HPSG are intended to serve as possible *pronoun antecendents.* Thus, if the sentences *Kim sees Sandy* and *She starts running* are spoken one after another then the pronoun *she* could refer to either *Kim* or *Sandy.* This also shows that set descriptions are an appropriate type of descriptions for encoding semantic indices since anaphora resolution then translates as an existential quantification over the *inds* role *i.e.* $\exists inds : PRN$ where PRN is a variable standing for a *pronoun.* Chapter 6 provides a set-value based formalisation of DRT.

Our next application for set-values is to treat subcategorisation items to be set-valued as opposed to list-valued which is the way they are treated in HPSG [Pollard & Sag 87, Pollard & Sag 92]. For instance, a partial representation for the lexical entry for the

verb *rely* (as in *John relies on Sandy*) will be:

$$(24) \qquad \left[\text{SYN|LOC}\left[\begin{array}{l}\text{HEAD|MAJ } V \\ \text{SUBCAT} \quad \{\, PP[-PRD, ON], NP[NOM, NORM] \,\}\end{array}\right]\right]$$

This lexical entry states that the verb *rely* takes two arguments namely a *non-predicative prepositional phrase headed by the preposition on* and a *nominative noun phrase.* The set-valued nature of the subcategorised items is intended to reflect our assumption that there is no *a priori* ordering on the subcategorised items.

Yet another important application is the use of set values in conjunction with linear precedence constraints (see Chapter 4 for its formalisation and applications).

An important property of set descriptions is that they are construed as partial descriptions in the sense that the set denoted by *Parking_Lot* in (25) (due to [Pollard & Moshier 90]) could denote either a parking lot containing two cars - one of which is a toyota model and the other is a 1984 model; or it could denote a parking lot containing a single car which is a 1984 toyota.

$$(25) \qquad Parking\_Lot = \left\{ Car\left[\; make \;:\; toyota \;\right],\; Car\left[\; year \;:\; 1984 \;\right] \right\}$$

The symbol Car denotes a primitive concept (or sort) symbol.

Now we are ready to deal with the formalisation of set descriptions.

3.3 The language $\mathcal{S}_1$

Firstly, in order to provide constraints for describing sets, we shall extend the language $\mathcal{L}_1$ with the following additional constraints:

$$x = f : \{x_1, \ldots, x_n\}, \qquad x = x_1 \sqcup \ldots \sqcup x_n, \qquad x \,\dot{\sqsubseteq}\, \geq f : n : \phi$$

where ϕ is a list of variables $x, y, z, \ldots$.

The constraint $x = f : \{x_1, \ldots, x_n\}$ models an object $\alpha(x)$ with at most n f-edges each leading to an individual $\alpha(x_i)$ as depicted in figure 3.6. The constraint $x = x_1 \sqcup \ldots \sqcup x_n$

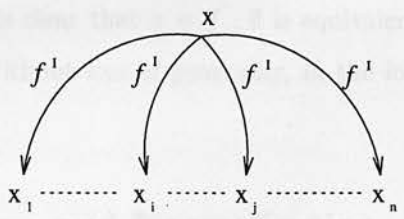


Figure 3.6: Modelling the constraint $x = f : \{x_1, \ldots, x_n\}$ as a graph

models the case when x is a disjunction of the n x_i's. And finally, the constraint $x \dot{\sqsubseteq} \geq f : n : \phi$ models the case when x has at least n f-edges and every $\alpha(x_i)$ such that x_i is in ϕ is connected via an f-edge from $\alpha(x)$.

In order to render a slightly simpler set of normalisation rules and to make the presentation easier, we shall assume that our normalisation rules are only applicable to constraint systems that have a certain *initial* form or to constraint systems that are derived *via* a finite number of application of our normalisation rules to a constraint system in an initial form.

Definition 3.3.1 *A constraint system C_s is an* ***initial*** *constraint system if $x \dot{\sqsubseteq} \geq f : n : \phi \in C_s$ implies that $\phi \equiv []$.*

We shall refer to the augmented language as $\mathcal{S}_1$.

We say that a relational algebra $\mathcal{I}$ and a variable assignment α satisfies the above constraints if the following conditions are satisfied:

1. $\mathcal{I}, \alpha \models x = f : \{x_1, \ldots, x_n\} \Longleftrightarrow f^I(\alpha(x)) = \{\alpha(x_1), \ldots, \alpha(x_n)\}$
2. $\mathcal{I}, \alpha \models x = x_1 \sqcup \ldots \sqcup x_n \Longleftrightarrow \alpha(x) = \alpha(x_i)$ for some x_i: $1 \leq i \leq n$.
3. $\mathcal{I}, \alpha \models x \dot{\sqsubseteq} \geq f : n : \phi$

   $\Longleftrightarrow$

   - $|f^I(\alpha(x))| \geq n$
   - for every y in ϕ: $(\alpha(x), \alpha(y)) \in f^I$
   - for every x_i, x_j in ϕ: $1 \leq i, j \leq \mathit{length}(\phi)$ such that $i \neq j$:

     $\alpha(x_i) \neq \alpha(x_j)$

From our definitions it is clear that $x = f : \emptyset$ is equivalent to $xf\uparrow$ and $x = f : \{y\}$ is equivalent to xfy. Without loss of generality, in the following we shall ignore the constraint $x = f : \emptyset$.

3.4 Normal Form and Normalisation

In this section we show that the language $\mathcal{S}_1$ admits a normal form and furthermore we shall show that the *relational graph algebra* (see Chapter 2) provides canonical interpretations for consistent $\mathcal{S}_1$ constraint systems in normal form. We then strengthen this claim by showing that every $\mathcal{S}_1$ constraint system can be transformed into normal form. This translation shows that the relational graph algebra provides canonical interpretations for every consistent $\mathcal{S}_1$ constraint system.

The following definitions are needed in the normal form and the normalisation rules:

- We say that a list of variables ϕ is **satisfiable** if there is no variable x such that x occurs more than *once* in ϕ otherwise ϕ is **unsatisfiable**. Note that if ϕ is *inconsistent* then the constraint $x \dot{\sqsubseteq} \geq f : n : \phi$ is *unsatisfiable*.

- The **f-successors** of x in a given constraint system C_s denoted by $succ(x, f)$ is given by the definition:

  - $succ(x, f) = \{y \mid xfy \in C_s \text{ or } x\ \exists f\ y \in C_s \text{ or } x = f : \{\ldots, y, \ldots\} \in C_s\}$

Definition 3.4.1 [Normal Form] *We say that a $\mathcal{S}_1$ constraint system C_s is in normal form if the following conditions are satisfied:*

1. *C_s satisfies all the conditions for normal form for the language $\mathcal{L}_1$ presented in Chapter 2*
2. *if $x = f : \{x_1, \ldots, x_n\} \in C_s$ then there are no constraints of the form $xFy \in C_s$ where F ranges over $f, \exists f, \forall f$.*
3. *the constraint $x = f : \{y\} \notin C_s$.*
4. *if $x = f : \{x_1, \ldots, x_n\} \in C_s$ then each of the x_i's (where $1 \leq i \leq n$) are distinct from each other.*
5. *there is at most one constraint of the form $x = f : \{x_1, \ldots, x_n\} \in C_s$ for each variable x.*

6. *if* $x = x_1 \sqcup \ldots \sqcup x_n \in C_s$ *then* $x \equiv x_i$ *for some* $x_i : 1 \leq i \leq n$.
7. *if* $x \dot{\sqsubseteq} \geq f : n : \phi \in C_s$ *such that* $n > 0$ *then the constraint* $x \; \forall f \; y \notin C_s$
8. *if* $x \dot{\sqsubseteq} \geq f : n : \phi \in C_s$ *and* ϕ *is satisfiable then:*
   - *for every variable* y *in* ϕ : $y \in succ(x, f)$
   - *if neither of* $xfy, x = f : \{x_1, \ldots, x_n\} \in C_s$ *then* $length(\phi) = n$ *and for every* y_i *in* ϕ : $x \; \exists f y_i \in C_s$.

As for the normal form definition for the language $\mathcal{L}_1$ the above normal form definition for $\mathcal{S}_1$ does not directly detect inconsistent constraint systems. For this purpose, we extend the definition of **clash** on a variable x as follows.

We say that a variable x contains a **clash** if either it contains a clash according to the definition of *clash* for the language $\mathcal{L}_1$ or if *any* of the following conditions is satisfied:

1. if $x \dot{\sqsubseteq} a, x = f : \{x_1, \ldots, x_n\} \in C_s$
2. if $x \dot{\sqsubseteq} \geq f : n : \phi \in C_s$ such that ϕ is unsatisfiable
3. if $x \dot{\sqsubseteq} \geq f : n : \phi, x = f : \{x_1, \ldots, x_m\} \in C_s$ such that $n > m$
4. if $x \dot{\sqsubseteq} \geq f : n : \phi, xfy \in C_s$ such that $n > 1$

Theorem 3.4.2 [Normal Form theorem] *If* C_s *is a* $\mathcal{S}_1$ *constraint system in normal form then:*

1. C_s *is inconsistent if there exists a variable* $x \in \mathcal{V}(C_s)$ *containing a clash*
2. *if* C_s *contains no variable containing a clash then* α *is the canonical solution for every* $x \in \mathcal{V}(C_s)$ *in the relational graph algebra* $\mathcal{R}$ *where:*
   - *for every variable* x *in* C_s:
     $\alpha(x) = (\xi(x), RG[con(x, C_s)])$ *where*
     (a) ξ *is an auxiliary function* $\xi : \mathcal{V} \longrightarrow \mathcal{V} \cup \mathcal{C}$ *defined by:*
     - *For each* x *such that* $x \dot{\sqsubseteq} c \in C_s$ *let* $\xi(x) = c$.
     - *For each* x *such that* $x \dot{\sqsubseteq} c \notin C_s$ *let* $\xi(x) = x$.

     (b) *and* $RG[C_G]$ *is a function that generates a graph from* C_G *defined by:*
     $RG[C_G] = C_{G3}$ *where*
     i. C_{G1} *is obtained from* C_G *by removing each constraint of the form* $x = f : \{x_1, \ldots, x_n\}$ *and replacing it by* $xfx_1, \ldots, xfx_n$.
     ii. C_{G2} *is a relational graph obtained from* C_{G1} *by removing each constraint of the form* $x \; \exists f \; y$ *and replacing it by* xfy.

iii. C_{G3} *is a relational graph obtained from* C_{G2} *by removing each constraint of the form* $x \dot{\sqsubseteq} c$ *and instead replacing every occurrence of* x *with* c.

- *for every constant symbol* c *in* C_s*:*
  - $\alpha(c) = \alpha(x)$ *if* $x \dot{\sqsubseteq} c \in C_s$
  - $\alpha(c) = (c, \emptyset)$ *if* $x \dot{\sqsubseteq} c \notin C_s$
- *for every atom* a *:* $\alpha(a) = (a, \emptyset)$
- *for every primitive concept symbol* C *in* C_s*:* $\alpha(C) = \{\alpha(x) \mid x \dot{\sqsubseteq} C \in C_s\}$

Proof: For the first part, assume that there exists a variable $x \in \mathcal{V}_s$ containing a clash then we know from the semantics of $\mathcal{S}_1$ constraints presented in the previous section together with the normal form definition 3.4.1 that there is no relational algebra that satisfies C_s.

For the second part, assume that C_s is consistent. Then we first need to show that $\alpha(x)$ is a solution for every $x \in \mathcal{V}(C_s)$. Let x be any variable in C_s.

1. if $x = f : \{x_1, \ldots, x_n\} \in C_s$ then we know from the definition of the normal form that there are no constraints of the form $xFy \in C_s$ where F ranges over $f, \exists f, \forall f$ and furthermore there is only one constraint of the form $x = f : \{x_1, \ldots, x_n\} \in C_s$. Secondly, since C_s is consistent it follows from the definition of *clash* that there are no constraints of the form $x \dot{\sqsubseteq} a$ or $x \dot{\sqsubseteq} \geq f : m : \phi \in C_s$ where $m > n$.

   Hence, $f^R(\alpha(x)) = \{\alpha(x_1), \ldots, \alpha(x_n)\}$ which means:
   $\mathcal{R}, \alpha \models x = f : \{x_1, \ldots, x_n\} \in C_s$.

2. if $x \dot{\sqsubseteq} \geq f : n : \phi, x = f : \{x_1, \ldots, x_m\} \in C_s$ then from the definition of *clash* we know that $m \geq n$ hence $|f^R(\alpha(x))| \geq m$.

   From the normal form definition we know that $x \; \exists f \; z \notin C_s$ and $xfy \notin C_s$.

   Again from the normal form definition, we know that for every $y_i \in \phi$: $y_i \in succ(x, f)$. Since $succ(x, f) = \{x_1, \ldots, x_n\}$, this implies that for every $y_i \in \phi$: $y_i \equiv x_j$ for some x_j $(1 \leq j \leq m)$.

   From the previous step, since $f^R(\alpha(x)) = \{\alpha(x_1), \ldots, \alpha(x_n)\}$, we conclude that $\mathcal{R}, \alpha \models x \dot{\sqsubseteq} \geq f : n : \phi$.

3. if $x \dot{\sqsubseteq} \geq f : n : \phi, xfy \in C_s$ then from the definition of *clash* we know that $n \leq 1$ and hence $\phi = [y]$ or $\phi = []$. Hence $\mid f^R(\alpha(x)) \mid = 1$ which means $\mathcal{R}, \alpha \models x \dot{\sqsubseteq} \geq f : n : \phi$.

4. if $x \dot{\sqsubseteq} \geq f : n : \phi, x \; \exists f \; y \in C_s$ then from the definition of *clash* we know that $\mid succ(x, f) \mid \geq n$. Furthermore, we know that for every y_i in ϕ there exists $x \; \exists f \; y_i \in C_s$. Hence, $\mid f^I(\alpha(x)) \mid \geq n$.

   This implies that $\mathcal{R}, \alpha \models x \dot{\sqsubseteq} \geq f : n : \phi$.

5. if $x \dot{\sqsubseteq} \geq f : n, x \; \forall f \; y \in C_s$ then from the normal form definition, we know that $n = 0$. Hence, $\mid f^I(\alpha(x)) \mid \uparrow$ *i.e.* $\mid f^I(\alpha(x)) \mid = 0$.

   Hence, $\mathcal{R}, \alpha \models x \dot{\sqsubseteq} \geq f : n : \phi$.

For every other constraint in C_s the normal form theorem for $\mathcal{L}_1$ holds.

Hence, this proves that $\mathcal{R}, \alpha \models C_s$ since it satisfies every constraint in C_s.

For the second part we also need to show that α is a canonical interpretation in the relational graph algebra $\mathcal{R}$. Assume that β is another assignment in $\mathcal{R}$ such that $\mathcal{R}, \beta \models C_s$. We shall then verify that the mapping:

$\gamma(\alpha(x)) = \beta(x)$ for every variable $x \in \mathcal{V}(C_s)$.

$\gamma(a^R) = a^R$ for every atom a.

is still a *partial endomorphism*.

We already know from the normal form theorem for the language $\mathcal{L}_1$ that γ is properly defined (*see* Chapter 2).

Next, to verify that γ retains the endomorphism property, we need to look at the following two cases.

1. if $(e_x, e_{x_i}) \in f^R$ such that $x = f : \{x_1, \ldots, x_n\} \in C_s$ then there exists $x_i \in \{x_1, \ldots, x_n\}$ such that $\alpha(x_i) = e_{x_i}$.

   Furthermore, $\mid f^I(\alpha(x)) \mid = n$ and since $\mathcal{R}, \beta \models x = f : \{x_1, \ldots, x_n\}$ we have $\mid f^I(\beta(x)) \mid \leq n$.

For each e_{x_i} such that $(e_x, e_{x_i}) \in f^R$ by the definition of γ we know that:

$$\gamma(\alpha(x_i)) = \gamma(e_{x_i}) = \beta(x_i)$$

Since the above is true for every $x_i \in \{x_1, \ldots, x_n\}$ which are mapped distinctly by α :

$$\text{for each } (e_x, e_{x_i}) \in f^R : (\gamma(e_{x_i}), \gamma(e_{x_i})) \in f^R$$

as required by the definition of *partial endomorphism* (see Chapter 2 section 2.2.1).

2. if $(e_x, e_{x_i}) \in f^R$ such that $\alpha(x) = e_x$ and $x \dot{\sqsubseteq} \geq f : n : \phi \in C_s$ then from the definition of the normal form we know that $|succ(x, f)| \geq n$ such that:

   - if $xfy \in C_s$ then $n = 1$
   - if $x = f : \{x_1, \ldots, x_m\} \in C_s$ then $m \geq n$
   - if $x \; \exists f \; y_i \in C_s$ then $length(\phi) = n$ and for each z_i in $\phi : x \; \exists f \; z_i \in C_s$

   We now need to show that the endomorphism property of γ holds in conjunction with each of the above constraints. But this is true from the normal form theorem for the language $\mathcal{L}_1$ and the previous case.

   This means that γ retains the endomorphism property in this case too.

Hence, α is a canonical interpretation in the relational graph algebra $\mathcal{R}$.

This result strengthens the case for the relational graph algebra as a model structure for providing canonical interpretations.

We are now ready to show that any $\mathcal{S}_1$ constraint system can be transformed into normal form and hence from the normal form theorem the consistency of any given constraint system can be decided.

3.5 Normalisation

The following is a complete set of normalisation rules for translating any given $\mathcal{S}_1$ constraint system into normal form. These rules are needed in addition to the norma-

lisation rules for the language $\mathcal{L}_1$ we described in Chapter 2. We assume that rule 7 is applied with the least priority among all the $\mathcal{S}_1$ and $\mathcal{L}_1$ rules to minimise the number of variables generated by this rule [1].

The following definition of *list union* is used in the normalisation rules:

> The **list union** $\phi_1 \cup \phi_2$ is defined as the list formed by combining the members of both ϕ_1 and ϕ_2 and removing any duplicates. In other words,
>
> - $[] \cup l = l$
> - $x.t \cup l = t \cup l$ if $x \in l$
> - $x.t \cup l = x.(t \cup l)$ if $x \notin l$

Normalisation rules for the language $\mathcal{S}_1$

1. $\{xFy, x = f : \{x_1, \ldots, x_n\}\} \cup C_s \longrightarrow \{xFy, y = x_1, \ldots, y = x_n\} \cup C_s$
   where F ranges over $f, \forall f$
2. $\{x\ \exists f\ y, x = f : \{x_1, \ldots, x_n\}\} \cup C_s \longrightarrow \{x = f : \{x_1, \ldots, x_n\}, y = x_1 \sqcup \ldots \sqcup x_n\} \cup C_s$
3. $\{x = f : \{y\}\} \cup C_s \longrightarrow \{xfy\} \cup C_s$
4. $\{x = f : \{x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n\}\} \cup C_s$
   $\longrightarrow$
   $\{x = f : \{x_1, \ldots, x_i, \ldots, \ldots, x_n\}\} \cup C_s$
   if $x_i \equiv x_j$
5. $\{x = f : \{x_1, \ldots, x_n\}, x = f : \{y_1, \ldots, y_m\}\} \cup C_s$
   $\longrightarrow$
   $\{x = f : \{x_1, \ldots, x_n\},$
   $x_1 = y_1 \sqcup \ldots \sqcup y_m,$
   $\ldots$
   $x_n = y_1 \sqcup \ldots \sqcup y_m,$
   $y_1 = x_1 \sqcup \ldots \sqcup x_n,$

[1] Note that rule 7 is the only rule among all the $\mathcal{S}_1$ and $\mathcal{L}_1$ rules that introduces new variables.

$\ldots$

$y_m = x_1 \sqcup \ldots \sqcup x_n\} \cup C_s$

where $n \leq m$

6. $\{x = x_1 \sqcup \ldots \sqcup x_n\} \cup C_s \longrightarrow \{x = x_1 \sqcup \ldots \sqcup x_n, x = x_i\} \cup C_s$

   if $1 \leq i \leq n$ and there is no $x_j, 1 \leq j \leq n$ such that $x = x_j \in C_s, x_j = x \in C_s$ or $x \equiv x_j$

7. $\{x \dot{\sqsubseteq} \geq f : n : \phi\} \cup C_s \longrightarrow \{x \dot{\sqsubseteq} \geq f : n : \phi.[y],\ x\ \exists f\ y\} \cup C_s$

   if $x = f : \{x_1, \ldots, x_m\} \notin C_s$, ϕ is *satisfiable*,

   $|\, succ(x, f)\,| < n$ and y is new

The condition $n \leq m$ in rule 5 is only an efficiency measure and is strictly speaking not needed. This additional condition ensures that the set constraint that is removed from the constraint system after the application of rule 5 is the larger one among the two.

Rule 6 is non-deterministic and introduces a n-ary choice point.

Rule 7 is required to fulfill the condition 8 of the normal form definition 3.4.1. This rule successively adds constraints of the form $x\ \exists f\ y$ to the constraint system until $|\, succ(x, f)\,|$ becomes equal to n. Since this rule is applied last, the introduction of new variables into the constraint system is delayed as much as possible.

Example: Consider the effect of applying the $\mathcal{S}_1$ normalisation rules on the following initial constraint system.

1. $C_{s0} = \{\underline{x = f : \{x_1,\ x_2,\ x_3\}},\ \underline{x = f : \{y_1,\ y_2,\ y_3\}},$
   $x_1\ \exists f\ z_a,\ x_2\ \exists f\ z_a,\ x_3\ \exists f\ z_b,$
   $y_1\ \exists f\ z_a,\ y_2\ \exists f\ z_b,\ y_3\ \exists f\ z_b,$
   $z_a \dot{\sqsubseteq} a,\ z_b \dot{\sqsubseteq} b\}$

2. Applying rule 5 we get:
   $C_{s1} = \{x = f : \{x_1,\ x_2,\ x_3\},$
   $\underline{x_1 = y_1 \sqcup y_2 \sqcup y_3}$

$$x_2 = y_1 \sqcup y_2 \sqcup y_3,$$
$$x_3 = y_1 \sqcup y_2 \sqcup y_3,$$
$$y_1 = x_1 \sqcup x_2 \sqcup x_3,$$
$$y_2 = x_1 \sqcup x_2 \sqcup x_3,$$
$$y_3 = x_1 \sqcup x_2 \sqcup x_3,$$
$$x_1\ \exists f\ z_a,\ x_2\ \exists f\ z_a,\ x_3\ \exists f\ z_b,$$
$$y_1\ \exists f\ z_a,\ y_2\ \exists f\ z_b,\ y_3\ \exists f\ z_b,$$
$$z_a \dot{\sqsubseteq} a,\ z_b \dot{\sqsubseteq} b\}$$

3. Adding $x_1 = y_1$ (rule 6) and rewriting y_1 to x_1 we get:

$C_{s2} = \{x = f : \{x_1,\ x_2,\ x_3\},$
$$x_1 = x_1 \sqcup y_2 \sqcup y_3,$$
$$\underline{x_2 = x_1 \sqcup y_2 \sqcup y_3},$$
$$x_3 = x_1 \sqcup y_2 \sqcup y_3,$$
$$x_1 = x_1 \sqcup x_2 \sqcup x_3,$$
$$y_2 = x_1 \sqcup x_2 \sqcup x_3,$$
$$y_3 = x_1 \sqcup x_2 \sqcup x_3,$$
$$\underline{x_1\ \exists f\ z_a},\ x_2\ \exists f\ z_a,\ x_3\ \exists f\ z_b,$$
$$\underline{x_1\ \exists f\ z_a},\ y_2\ \exists f\ z_b,\ y_3\ \exists f\ z_b,$$
$$z_a \dot{\sqsubseteq} a,\ z_b \dot{\sqsubseteq} b\}$$

4. Notice that rule 6 is now *not* applicable to the constraint $x_1 = x_1 \sqcup x_2 \sqcup x_3$, an efficient side effect of the previous rule application. We first eliminate $x_1\ \exists f\ z_a$ by using the $\mathcal{L}_1$ normalisation rules and then we apply rule 6 again to rewrite x_2 by x_1 to get:

$C_{s3} = \{\underline{x = f : \{x_1,\ x_1,\ x_3\}},$
$$x_1 = x_1 \sqcup y_2 \sqcup y_3,$$
$$x_1 = x_1 \sqcup y_2 \sqcup y_3,$$
$$\underline{x_3 = x_1 \sqcup y_2 \sqcup y_3},$$
$$x_1 = x_1 \sqcup x_1 \sqcup x_3,$$
$$y_2 = x_1 \sqcup x_1 \sqcup x_3,$$
$$y_3 = x_1 \sqcup x_1 \sqcup x_3,$$

$x_1 \; \exists f \; z_a, \; x_3 \; \exists f \; z_b,$

$y_2 \; \exists f \; z_b, \; y_3 \; \exists f \; z_b,$

$z_a \; \dot{\sqsubseteq} \; a, \; z_b \; \dot{\sqsubseteq} \; b\}$

5. We then simplify $x = f : \{x_1, \; x_1, \; x_3\}$ to $x = f : \{x_1, \; x_3\}$ (rule 4). Next, we eliminate x_3 by rewriting to x_1 (rule 6) to get:

$C_{s4} = \{x = f : \{x_1, \; x_3\},$

$x_1 = x_1 \sqcup y_2 \sqcup y_3,$

$x_1 = x_1 \sqcup y_2 \sqcup y_3,$

$\underline{x_1 = x_1 \sqcup y_2 \sqcup y_3},$

$x_1 = x_1 \sqcup x_1 \sqcup x_3,$

$y_2 = x_1 \sqcup x_1 \sqcup x_3,$

$y_3 = x_1 \sqcup x_1 \sqcup x_3,$

$x_1 \; \exists f \; z_b,$

$y_2 \; \exists f \; z_b, \; y_3 \; \exists f \; z_b,$

$z_a \; \dot{\sqsubseteq} \; a, \; z_b \; \dot{\sqsubseteq} \; b\}$

6. Further simplification by rewriting z_a to z_b (rule 6) causes a **clash** since we we get $z_b \; \dot{\sqsubseteq} \; b$ and $z_b \; \dot{\sqsubseteq} \; a$. Hence we take an alternative choice point by rewriting x_3 to y_2 instead. This gives:

$C_{s5} = \{x = f : \{x_1, \; y_2\},$

$x_1 = x_1 \sqcup y_2 \sqcup y_3,$

$x_1 = x_1 \sqcup y_2 \sqcup y_3,$

$y_2 = x_1 \sqcup y_2 \sqcup y_3,$

$x_1 = x_1 \sqcup x_1 \sqcup y_2,$

$y_2 = x_1 \sqcup x_1 \sqcup y_2,$

$\underline{y_3 = x_1 \sqcup x_1 \sqcup y_2},$

$x_1 \; \exists f \; z_a, \; y_2 \; \exists f \; z_b,$

$y_2 \; \exists f \; z_b, \; y_3 \; \exists f \; z_b,$

$z_a \; \dot{\sqsubseteq} \; a, \; z_b \; \dot{\sqsubseteq} \; b\}$

7. We can then eliminate $y_2 \; \exists f \; z_b$ by using the feature elimination rules for $\mathcal{L}_1$. We then rewrite y_3 by x_1 (rule 6) but this again gives a clash. The only choice

remaining is to rewrite y_3 by y_2 to get the following after simplification:

$$
\begin{aligned}
C_{s6} = \{ & x = f : \{x_1,\ y_2\},\\
& x_1 = x_1 \sqcup y_2 \sqcup y_2,\\
& x_1 = x_1 \sqcup y_2 \sqcup y_2,\\
& y_2 = x_1 \sqcup y_2 \sqcup y_2,\\
& x_1 = x_1 \sqcup x_2 \sqcup y_2,\\
& y_2 = x_1 \sqcup x_2 \sqcup y_2,\\
& y_2 = x_1 \sqcup x_2 \sqcup y_2,\\
& x_1\ \exists f\ z_a,\\
& y_2\ \exists f\ z_b,\\
& z_a \dot{\sqsubseteq} a,\ z_b \dot{\sqsubseteq} b\}
\end{aligned}
$$

8. C_{s6} is in normal form and the normalisation procedure terminates.

3.5.1 Invariance, Termination and Completeness

Lemma 3.5.1 [Invariance of rule 5] *Rule 5 is equivalence preserving.*

Proof: We show that if C'_s is derived from C_s by applying rule 5 then for any relational algebra $\mathcal{I}$ and a $\mathcal{I}$-assignment $\alpha : \mathcal{I}, \alpha \models C_s \Longleftrightarrow \mathcal{I}, \alpha \models C'_s$

Assume that there exists a relational algebra $\mathcal{I}$ and a $\mathcal{I}$-assignment α such that $\mathcal{I}, \alpha \models x = f : \{x_1, \ldots, x_n\}$ and $\mathcal{I}, \alpha \models x = f : \{y_1, \ldots, y_m\}$ where $n \leq m$.

From the definition of satisfaction for set constraints, we know that:

$f^I(\alpha(x)) = \{\alpha(x_1), \ldots, \alpha(x_n)\}$, $f^I(\alpha(x)) = \{\alpha(y_1), \ldots, \alpha(y_m)\}$

and

$|f^I(\alpha(x))| \leq n$.

This means that:

(26)
- for each $x_i : 1 \leq i \leq n$ there exists $y_j : 1 \leq j \leq m$ such that $\alpha(x_i) = y_j$
- for each $y_i : 1 \leq i \leq m$ there exists $x_j : 1 \leq j \leq n$ such that $\alpha(x_i) = y_j$

But, this is exactly what the following constraints express.

(27) $x_1 = y_1 \sqcup \ldots \sqcup y_m,$

$\ldots$

$x_n = y_1 \sqcup \ldots \sqcup y_m,$

$y_1 = x_1 \sqcup \ldots \sqcup x_n,$

$\ldots$

$y_m = x_1 \sqcup \ldots \sqcup x_n$

where $n \leq m$

Conversely, if $\mathcal{I}, \alpha$ satisfies the right hand side of rule 5 then we need to show that:
$\mathcal{I}, \alpha \models x = f : \{y_1, \ldots, y_m\}$.

Firstly, since $\mathcal{I}, \alpha \models x = f : \{x_1, \ldots, x_n\}$ which means:
$f^I(\alpha(x)) = \{\alpha(x_1), \ldots, \alpha(x_n)\}$.

Secondly, since $\mathcal{I}, \alpha$ satisfies the disjuncts on the right hand side, it follows that $\mathcal{I}, \alpha$ satisfies the conditions in 26. But, this means:
$f^I(\alpha(x)) = \{\alpha(y_1), \ldots, \alpha(y_m)\}$.

Hence, $\mathcal{I}, \alpha \models x = f : \{y_1, \ldots, y_m\}$.

Since, rule 5 preserves the semantics of every variable and does not introduce any new variables, rule 5 is equivalence preserving.

The notion of invariance we adopt for the normalisation rules of $\mathcal{S}_1$ is a weaker one as compared to the notion of invariance adopted for the language $\mathcal{L}_1$. This is due to the fact that $\mathcal{S}_1$ provides disjunctions and hence equivalence is not preserved by the normalisation rules.

Theorem 3.5.2 [Invariance] *C_s is a consistent constraint system iff at least one instance of every applicable rule transforms C_s into a consistent constraint system.*

Proof: For ths first part, assume that the original constraint system is inconsistent, then we need to show that the application of any of the normalisation rule does not transform the original constraint system into a consistent one. This is quite easy to verify since each rule preserves inconsistency. For rule 6 if $x = x_1 \sqcup \ldots \sqcup x_n$ is inconsistent then every application of rule 6 results in an inconsistent constraint.

For the second part, it is enough to note that every normalisation rule apart from rule 6 preserves equivalence. Rule 6 on the other hand preserves consistency in at least one of its instances among the n possible choices. This means that the application of rule 6 could possibly transform a consistent constraint system into an inconsistent one. But, there would be at least one instance when rule 6 makes the right choice (*i.e.* a consistency preserving choice). The fact that this is indeed the case is guaranteed by the semantics of the constraint $x = x_1 \sqcup \ldots \sqcup x_n$.

This proves the theorem.

Actually a slightly stronger claim can be made namely that if C_s is a consistent constraint system, $\mathcal{I}$ an relational algebra and α an $\mathcal{I}$-assignment such that $\mathcal{I}, \alpha \models C_s$ then there is at least one instance of the application of the normalisation rules which transform C_s to C_s' such $\mathcal{I}, \alpha \models C_s'$. Since every rule apart from rule 6 which deals with disjunctions is equivalence perserving it is enough to note that this claim is valid for rule 6.

Theorem 3.5.3 [Termination] *There is no infinite chain issuing from the application of the normalisation rules.*

Proof: To prove the termination claim we extend the complexity measure on multiset orderings [Dershowitz & Manna 78] on $\mathcal{L}_1$ constraint systems as defined in Chaper 2 to $\mathcal{S}_1$ constraint systems by:

$| \; x = x_1 \sqcup \ldots \sqcup x_n \; | = 0$ if $x = x_i \in C_s$, $x_i = x \in C_s$ or $x \equiv x_i$ for some $x_i : 1 \leq i \leq n$

$| \; x = x_1 \sqcup \ldots \sqcup x_n \; | = 4$ otherwise

$| \; x = f : \{x_1, \ldots, x_n\} \; | = 8$

$| \; x \dot{\sqsubseteq} \geq f : n : \phi \; | = 0$
if ϕ is unsatisfiable

$| \; x \dot{\sqsubseteq} \geq f : n : \phi \; | = 0$
if $x = f : \{x_1, \ldots, x_m\} \in C_s$

$| x \dot{\sqsubseteq} \geq f : n : \phi | = (n + 1 - length(\phi)) \times 10$

if ϕ is satisfiable and $x = f : \{x_1, \ldots, x_m\} \notin C_s$

It is easy to see that the complexity measure $K(C_s)$ associated with each constraint system C_s (see Chapter 2) decreases with the application of each normalisation rule.

Theorem 3.5.4 [Completeness of Rewriting] *In every completion of the normalisation rules any $\mathcal{S}_1$ constraint system is transformed into normal form.*

Proof: To prove the completeness claim, we need to ensure that if a constraint system is not in normal form then one of the normalisation rules apply. For each of the conditions for normal form given in definition 3.4.1, it is easy to see that there is a corresponding rule. Hence together with the termination claim, we have the theorem.

3.6 Conclusions

In the previous sections we have shown that constraints on sets can be conveniently handled by extending the language $\mathcal{L}_1$ with 3 additional constructs which are:

$$x = f : \{x_1, \ldots, x_n\}, \qquad x = x_1 \sqcup \ldots \sqcup x_n, \qquad x \dot{\sqsubseteq} \geq f : n : \phi$$

However, to turn $\mathcal{S}_1$ into a practical knowledge representation language, *path equations* that we considered in Chapter 2 have to be added. This however does not introduce any complications since we have already shown that *path equations* can be eliminated in linear time as they can be considered a compact representation for a set of feature constraints which hides variables that are not of prime concern for the application developer.

We believe that the language $\mathcal{S}_1$ augmented with path equations can be considered as a powerful alternative to constraint languages such as PATR-II. Such an augmented language will be suitable for applications that do not require the full power of a term description language that provides set descriptions but at the same time cannot be handled by the limited representation services provided by PATR-II.

Note, that since $\mathcal{S}_1$ provides primitive support for disjunctions it can also be considered

to be an alternative to representation systems that augment PATR-II with disjunctions such as D-PATR [Karttunen 86a].

However constraint based grammatical frameworks such as HPSG are based on the idea of employing a term language for representing linguistic knowledge. Hence term description languages deserves study.

This discussion provides the motivation for studying the term description language $\mathcal{ALS}$ which will provide terms for so called partial description of sets. This is what we shall do next.

3.7 The term description language $\mathcal{ALS}$

In this section we extend the term description language $\mathcal{ALV}$ that we developed in Chapter 2 by adding *set descriptions.*

The language $\mathcal{ALS}$ is the language $\mathcal{ALV}$ with one extra construct namely the *set description* which has the form:

$$f : \{T_1, \ldots, T_n\}$$

Given a relational algebra $\mathcal{I}$ and a variable assignment α, the denotation function $.^{\mathcal{I},\alpha}$ is extended by:

$$[\![f : \{T_1, \ldots, T_n\}]\!]^{\mathcal{I},\alpha} =$$
$$\{e \in \mathcal{U}^I \mid f^I(e) = \{e_1, \ldots, e_n\} \wedge e_1 \in [\![T_1]\!] \wedge \ldots \wedge e_n \in [\![T_n]\!]\}$$

Corollary 3.7.1 *For any* $e \in [\![f : \{T_1, \ldots, T_n\}]\!]^{\mathcal{I},\alpha} \mid f^I(e) \mid \leq n$.

Intuitively, $f : \{T_1, \ldots, T_n\}$ denotes the set of all elements which have at most n f-edges that leads to the set denoted by each T_i.

Definition 3.7.2 *We say that terms* S *and* T *in* $\mathcal{ALS}$ *are equivalent, written* $S \equiv T$, *if for every relational algebra* $\mathcal{I}$ *and a* $\mathcal{I}$*-assignment* α : $[\![S]\!]^{\mathcal{I},\alpha} = [\![T]\!]^{\mathcal{I},\alpha}$.

It follows from this that while $\{a,\ b,\ b\}$ is equivalent to $\{a,\ b\}$ where $a, b \in \mathcal{A}t$, it does not follow that $\{\exists f : a,\ \exists f : b,\ \exists f : b\}$ and $\{\exists f : a,\ \exists f : b\}$ are equivalent.

3.7.1 $\mathcal{ALS}$ and Concept languages with Number restrictions

In this section, we relate $\mathcal{ALS}$ with concept languages containing the so called number restrictions. For this purpose, we first extend the language with some extra constructs which are known as *number restrictions* [Hollunder & Nutt 90]:

$$\geq f : n, \qquad \leq f : n$$

where $f \in \mathcal{F}$ is a relation symbol and $n \geq 0$ is a natural number.

Given a relational algebra $\mathcal{I}$ and a $\mathcal{I}$-assignment α the denotation of these two constructs is provided by the following definitions:

1. $[\![\geq f : n]\!]^{\mathcal{I},\alpha} = \{e \in \mathcal{U}^I \mid |f^I(e)| \geq n\}$
2. $[\![\leq f : n]\!]^{\mathcal{I},\alpha} = \{e \in \mathcal{U}^I \mid |f^I(e)| \leq n\}$

Theorem 3.7.3 *The following equivalences relate the term* $\geq f : n$ *with the term* $\leq f : n$*:*

1. $\neg \leq f : n \equiv \geq f : n+1$
2. $\neg \geq f : n+1 \equiv \leq f : n$

Proof:

1. (a) $[\![\neg \leq f : n]\!]^{\mathcal{I},\alpha} = \mathcal{U}^I - \{e \in \mathcal{U}^I \mid |f^I(e)| \leq n\}$
   (b) $= \{\{e \in \mathcal{U}^I \mid |f^I(e)| > n\}$
   (c) $= \{\{e \in \mathcal{U}^I \mid |f^I(e)| \geq n+1\}$
   (d) $= [\![\geq f : n+1]\!]^{\mathcal{I},\alpha}$
2. Negating both sides of the previous result we get the desired result.

We now demonstrate that set descriptions can be equivalently encoded by number restrictions.

Theorem 3.7.4 *The following equivalences relate number restrictions with set descriptions:*

1. $f : \{\underbrace{\top, \ldots, \top}_{n \text{ times}}\} \equiv \leq f : n \sqcap \exists f : \top$
2. $\leq f : n \equiv f : \{\underbrace{\top, \ldots, \top}_{n \text{ times}}\} \sqcup \forall f : \bot$
3. $\neg f : \{\underbrace{\top, \ldots, \top}_{n \text{ times}}\} \equiv \geq f : n+1 \sqcup \forall f : \bot$
4. $\geq f : n+1 = \neg f : \{\underbrace{\top, \ldots, \top}_{n \text{ times}}\} \sqcap \exists f : \top$
5. $f : \{T_1, \ldots, T_n\} \equiv \leq f : n \sqcap \exists f : T_1 \sqcap \ldots \sqcap \exists f : T_n \sqcap \forall f : (T_1 \sqcup \ldots \sqcup T_n)$
6. $\neg f : \{T_1, \ldots, T_n\} \equiv$
   $\geq f : n+1 \sqcup \forall f : \neg T_1 \sqcup \ldots \sqcup \forall f : \neg T_n \sqcup \exists f : (\neg T_1 \sqcap \ldots \sqcap \neg T_n)$

Proof: Let $\mathcal{I}$ be any relational algebra and let α be an $\mathcal{I}$-assignment.

1. (a) i. For any $e \in [\![f : \{\underbrace{\top, \ldots, \top}_{n \text{ times}}\}]\!]^{\mathcal{I},\alpha}$, $f^I(e)\downarrow$ and $|f^I(e)| \leq n$.

   Hence $e \in [\![\leq f : n \sqcap \exists f : \top]\!]^{\mathcal{I},\alpha}$.

   (b) i. For any $e \in [\![\leq f : n \sqcap \exists f : \top]\!]^{\mathcal{I},\alpha}$, $e \in [\![\leq f : n]\!]^{\mathcal{I},\alpha}$ and $e \in [\![\exists f : \top]\!]^{\mathcal{I},\alpha}$.

   Hence $f^I(e)\downarrow$ and $|f^I(e)| \leq n$.

   ii. This means, $e \in [\![f : \{\underbrace{\top, \ldots, \top}_{n \text{ times}}\}]\!]^{\mathcal{I},\alpha}$.

   Hence, $f : \{\underbrace{\top, \ldots, \top}_{n \text{ times}}\} \equiv \leq f : n \sqcap \exists f : \top$

2. (a) Since, $f : \{\underbrace{\top, \ldots, \top}_{n \text{ times}}\} \equiv \leq f : n \sqcap \exists f : \top$.

   (b) This means:

   $f : \{\underbrace{\top, \ldots, \top}_{n \text{ times}}\} \sqcup \neg \exists f : \top \equiv (\leq f : n \sqcap \exists f : \top) \sqcup \neg \exists f : \top$

   $\equiv (\leq f : n \sqcup \neg \exists f : \top) \sqcap (\exists f : \top \sqcup \neg \exists f : \top)$

   $\equiv (\leq f : n \sqcup \neg \exists f : \top) \sqcap \top$

   $\equiv \leq f : n \sqcup \forall f : \bot$

   (c) Since $\forall f : \bot \sqsubseteq \leq f : n$, we get:

   $\leq f : n \equiv f : \{\underbrace{\top, \ldots, \top}_{n \text{ times}}\} \sqcup \forall f : \bot$

3. (a) Since $f : \{\underbrace{\top, \ldots, \top}_{n \text{ times}}\} \equiv \leq f : n \sqcap \exists f : \top$,

   $\neg f : \{\underbrace{\top, \ldots, \top}_{n \text{ times}}\} \equiv \neg (\leq f : n \sqcap \exists f : \top)$

$\equiv (\geq f : n+1 \sqcap \exists f : \top) \sqcup \neg \exists f : \top$

$\equiv (\geq f : n+1 \sqcup \forall f : \bot) \sqcap (\exists f : \top \sqcup \neg \exists f : \top)$

$\equiv (\geq f : n+1 \sqcup \forall f : \bot) \sqcap \top$

(b) Hence $\neg f : \underbrace{\{\top, \ldots, \top\}}_{n \text{ times}} \equiv \geq f : n+1 \sqcup \forall f : \bot$

4. (a) Taking into account the previous result:

$\neg f : \underbrace{\{\top, \ldots, \top\}}_{n \text{ times}} \sqcap \exists f : \top \equiv (\geq f : n+1 \sqcup \forall f : \bot) \sqcap \exists f : \top$

$\equiv (\geq f : n+1 \sqcap \exists f : \top) \sqcup (\forall f : \bot \sqcap \exists f : \top)$

$\equiv (\geq f : n+1 \sqcap \exists f : \top) \sqcup \bot$

$\equiv \geq f : n+1 \sqcap \exists f : \top$

(b) Since for every $e \in [\![\geq f : n+1]\!]^{\mathcal{I},\alpha}$, $e \in [\![\exists f : \top]\!]^{\mathcal{I},\alpha}$,

it follows that $[\![\geq f : n+1]\!]^{\mathcal{I},\alpha} \subseteq [\![\exists f : \top]\!]^{\mathcal{I},\alpha}$.

Hence $\geq f : n+1 \sqcap \exists f : \top \equiv \geq f : n+1$

(c) Hence $\geq f : n+1 \equiv \neg f : \underbrace{\{\top, \ldots, \top\}}_{n \text{ times}} \sqcap \exists f : \top$

5. (a) Let $(\{T_1, \ldots, T_n\})^\delta \equiv \leq f : n \sqcap \exists f : T_1 \sqcap \ldots \sqcap \exists f : T_n \sqcap \forall f : (T_1 \sqcup \ldots \sqcup T_n)$

(b) Let $e \in [\![\{T_1, \ldots, T_n\}]\!]^{\mathcal{I},\alpha}$,

then $f^I(e) = \{(e, e_1), \ldots, (e, e_n)\}$

where $e_1 \in [\![T_1]\!]^{\mathcal{I},\alpha}, \ldots, e_n \in [\![T_n]\!]^{\mathcal{I},\alpha}$

(c) Hence, $e \in [\![\leq f : n]\!]^{\mathcal{I},\alpha}$,

$e \in [\![\exists f : T_1]\!]^{\mathcal{I},\alpha}, \ldots, e \in [\![\exists f : T_n]\!]^{\mathcal{I},\alpha}$ and

$e \in [\![\forall f : (T_1 \sqcup \ldots \sqcup T_n)]\!]^{\mathcal{I},\alpha}$

(d) Hence $\{T_1, \ldots, T_n\} \sqsubseteq (\{T_1, \ldots, T_n\})^\delta$

(e) On the other hand, if $e \in [\![(\{T_1, \ldots, T_n\})^\delta]\!]^{\mathcal{I},\alpha}$ then all the conditions in step 5c hold.

(f) Taking into account the fact that $|f^I(e)| \leq n$,

let $f^I(e) = \{(e, x_1), \ldots, (e, x_n)\}$

where we repeat some of the elements such that there are exactly n of them for consideration.

Then the following conditions hold:

- $\exists x_1 \in [\![T_1]\!]^{\mathcal{I},\alpha}$ such that $(e, x_1) \in f^I(e)$, ... , $\exists x_n \in [\![T_n]\!]^{\mathcal{I},\alpha}$ such that $(e, x_n) \in f^I(e)$
- for every $(e, x_i) \in f^I(e) : e_i \in [\![T_i]\!]^{\mathcal{I},\alpha}$ for some $i, 1 \leq i \leq n$

(g) Hence $e \in [\![\{T_1, \ldots, T_n\}]\!]^{\mathcal{I},\alpha}$

(h) Hence $\{T_1, \ldots, T_n\} \sqsubseteq (\{T_1, \ldots, T_n\})^\delta$

(i) Taking into account step 4.4.1 this means: $\{T_1, \ldots, T_n\} \equiv (\{T_1, \ldots, T_n\})^\delta$

6. By negating both sides of the previous result we get the desired result.

These results show that set descriptions can be equivalently encoded by number restrictions which can be considered to be a more primitive form of description as compared to set descriptions. However, if set descriptions are to be completely eliminated then we will lose the benefit of a compact representation since as shown by the above result expanding set descriptions with number restrictions introduces a large number of concept descriptions. Thus from a practical point of view it is more beneficial to provide set descriptions. This argument can be strengthened by observing that for natural language applications [Pollard & Sag 87, Pollard & Sag 92, Rounds 88] set descriptions are a natural representation structure.

The above argument for providing set descriptions turns out to be further justified since we shall see in the following sections that consistency testing of both number restrictions and set descriptions can be achieved by employing the constraint solving machinery available in $\mathcal{S}_1$. This leads to the fact that preserving the compact representation offered by set descriptions lends itself to a more efficient constraint solving machinery which would otherwise be lost.

3.8 Consistency checking of $\mathcal{ALS}$ terms

The following theorem demonstrates that consistency checking of set descriptions can be achieved quite efficiently without breaking them down into constraints containing number restrictions.

Theorem 3.8.1 *The constraint system* $C_s = \{x \doteq f : \{T_1, \ldots, T_n\}\}$ *is consistent iff the constraint system:*

$$C'_s = \{x = f : \{x_1, \ldots, x_n\}, x_1 \doteq T_1, \ldots, x_n \doteq T_n\}$$

where $x_1, \ldots, x_n$ *are new, is consistent.*

Proof: For the first part assume that C_s is consistent then it follows that there exists a relational algebra $\mathcal{I}$ and a $\mathcal{I}$-assignment α such that $e \in [\![f : \{T_1, \ldots, T_n\}]\!]^{\mathcal{I},\alpha}$ such that $f^I(e) = \{e_1, \ldots, e_n\}$ where $e_1 \in [\![T_1]\!]^{\mathcal{I},\alpha}, \ldots, e_n \in [\![T_n]\!]^{\mathcal{I},\alpha}$.

This means that we can define an $\mathcal{I}$ assignment β such that $\beta(x_1) = e_1, \ldots, \beta(x_n) = e_n$. But this means that C'_s is consistent.

On the other hand, assume that C'_s is consistent. This means that:
$\alpha(x_1) \in [\![T_1]\!]^{\mathcal{I},\alpha}, \ldots, \alpha(x_n) \in [\![T_n]\!]^{\mathcal{I},\alpha}$.
Hence, $f^I(\alpha(x)) = \{\alpha(x_1), \ldots, \alpha(x_n)\}$.

This means that $\alpha(x) \in [\![f : \{T_1, \ldots, T_n\}]\!]^{\mathcal{I},\alpha}$.

Hence, C_s is consistent.

Corollary 3.8.2 *The constraint system* $C_s = \{x \doteq X, X \doteq f : \{X_1, \ldots, X_n\}\}$ *is consistent iff the constraint system:*

$$C'_s = \{x \doteq X, X \doteq f : \{X_1, \ldots, X_n\}, x = f : \{x_1, \ldots, x_n\}, x_1 \doteq X_1, \ldots, x_n \doteq X_n\}$$

where $x_1, \ldots, x_n$ *are new, is consistent.*

3.8.1 The language $\mathcal{S}_2$

In this section we consider augmentations to the language $\mathcal{L}_2$ to enable consistency testing of $\mathcal{ALS}$ terms. Let $\mathcal{S}_2$ be the constraint language obtained by including:

- every $\mathcal{L}_2$ constraint
- every $\mathcal{S}_1$ constraint
- and constraints having the following forms:
  $\overline{X} \doteq f : \{T_1, \ldots, T_n\}$, $\overline{X} \doteq \neg f : \{T_1, \ldots, T_n\}$, $X \doteq \geq f : n$, $x \doteq \geq f : n : \phi$
  where $T_1, \ldots, T_n$ are $\mathcal{ALS}$ terms

Next, we need to extend the definition of *path* between big variables (see Chapter 2) to take into account the above additional constraints.

Definition 3.8.3 [Path] *Given a constraint system C_s the relation $\underset{P}{\rightarrow}$ that relates two big variables and a path P is inductively defined as the least relation satisfying:*

1. $X \underset{[]}{\rightarrow} X$
2. $X \underset{[Y]}{\rightarrow} Y$ *if* $X \dot{\sqsubseteq} \exists f : Y \in C_s$ *or* $X \dot{\sqsubseteq} \forall f : Y \in C_s$
3. $X \underset{[\sqcup]}{\rightarrow} Y$ *if* $X \dot{\sqsubseteq} Y \sqcup \overline{Z} \in C_s$
4. $X \underset{[\sqcup]}{\rightarrow} Z$ *if* $X \dot{\sqsubseteq} \overline{Y} \sqcup Z \in C_s$
5. $X \underset{[Y_i]}{\rightarrow} Y_i$ *where* $1 \leq i \leq n$ *if* $X \dot{\sqsubseteq} f : \{\ldots, Y_i, \ldots\} \in C_s$
6. $X \underset{P \bullet Q}{\rightarrow} Z$ *if* $X \underset{P}{\rightarrow} Y$ *and* $Y \underset{Q}{\rightarrow} Z$

where the operation $\bullet$ is the concatenation *operation over strings.*

We shall refer to the augmented language as $\mathcal{S}_2$. Furthermore, we assume that the *acyclicity condition* applies to $\mathcal{S}_2$ constraint systems.

3.8.2 Normal Form

Now, we are ready to introduce the normal form for the language $\mathcal{S}_2$.

Definition 3.8.4 [Normal Form] *We say that a $\mathcal{S}_2$ constraint system C_s is in normal form if C_s satisfies the following conditions:*

1. *C_s obeys all the conditions for normal form for the language $\mathcal{L}_2$ provided in Chapter 2*
2. *C_s obeys all the conditions for normal form for the language $\mathcal{S}_1$ provided in definition 3.4.1*
3. *the constraint $\overline{X} \dot{\sqsubseteq} \neg f : \{\Gamma_1, \ldots, \Gamma_n\} \notin C_s$*
4. *if $X \dot{\sqsubseteq} f : \{\Gamma_1, \ldots, \Gamma_n\} \in C_s$ then each Γ_i $(1 \leq i \leq n)$ is a variable $\overline{X}_i$*
5. *the constraint $x \dot{\sqsubseteq} f : \{\Gamma_1, \ldots, \Gamma_n\} \notin C_s$*
6. *if $x = f : \{x_1, \ldots, x_n\}, \overline{X} \dot{\sqsubseteq} \forall f : \overline{Y} \in C_s$ such that*
   - either $x \dot{\sqsubseteq} \overline{X} \in C_s$
   - or $x \equiv \overline{X}$

then for each $x_i : 1 \leq i \leq n$

- either $x_i \equiv \overline{Y}$
- or $\overline{Y} \equiv Y$ *and* $x_i \dot{\sqsubseteq} Y \in C_s$

7. *if* $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} f : \{\overline{X}_1, \ldots, \overline{X}_n\} \in C_s$ *then there exists* $x = f : \{x_1, \ldots, x_m\} \in C_s$ *such that* $m \leq n$ *and the following conditions hold:*
   - *for each* $\overline{X}_i : 1 \leq i \leq n$ *there exists* $x_j : 1 \leq j \leq m$ *such that either* $x_j \dot{\sqsubseteq} \overline{X}_i \in C_s$ *or* $x_j \equiv \overline{X}_i$ *and*
   - *for each* $x_j : 1 \leq j \leq m$ *there exists* $\overline{X}_i : 1 \leq i \leq n$ *such that either* $x_j \dot{\sqsubseteq} X_i \in C_s$ *or* $x_j \equiv \overline{X}_i$
8. *if* $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \geq f : n \in C_s$ *then there exists* $x \dot{\sqsubseteq} \geq f : n : \phi \in C_s$

We say that a variable x in a $\mathcal{S}_2$ constraint system C_s contains a **clash** if either of the following conditions hold:

- x contains a *clash* according to the definition of *clash* for the the language $\mathcal{S}_1$
- x contains a *clash* according to the definition of *clash* for the the language $\mathcal{L}_2$.

Theorem 3.8.5 [Normal Form] *Every* $\mathcal{S}_2$ *constraint system in normal form not containing a clash is consistent.*

Proof: We shall demonstrate that the interpretation $\mathcal{R}, \alpha$ constructed by the normal form theorem for the language $\mathcal{S}_1$ extends to $\mathcal{S}_2$ constraint systems.

First of all, it is clear that if C_s contains a clash then C_s is inconsistent.

On the hand, to show that $\mathcal{R}, \alpha \models C_s$ if C_s does not contain a clash, first we need to provide interpretations for big variables as we have done in the normal form theorem for the language $\mathcal{L}_2$ (in Chapter 2).

Let $\alpha(X)$ be defined by:
$\alpha(X) = \{\alpha(x) \mid x \dot{\sqsubseteq} X \in C_s\}$

This means that $\mathcal{R}, \alpha \models x \dot{\sqsubseteq} X$ for every constraint of the form $x \dot{\sqsubseteq} X$.

From the normal form theorem for the language $\mathcal{S}_1$ we know that $\mathcal{R}, \alpha \models c$ for every x-constraint c. Secondly, from the normal form theorem for the language $\mathcal{L}_2$ we know that the above interpretation extends to $\mathcal{L}_2$ constraint systems. Hence, it is enough

to consider the following additional cases not considered by the normal form theorem for the language $\mathcal{L}_2$ which deal with the interaction between $\mathcal{S}_1$ constraints and $\mathcal{L}_1$ constriants:

1. if $x = f : \{x_1, \ldots, x_n\}, \overline{X} \dot{\sqsubseteq} \forall f : \overline{Y} \in C_s$ such that
   - *either* $x \dot{\sqsubseteq} \overline{X} \in C_s$
   - *or* $x \equiv \overline{X}$

   then we know from the normal form definition 3.8.4 that for each $x_i : 1 \leq i \leq n$
   - *either* $x_i \equiv \overline{Y}$
   - *or* $\overline{Y} \equiv Y$ and $x_i \dot{\sqsubseteq} Y \in C_s$

   This means that for each $x_i : 1 \leq i \leq n$:
   $\alpha(x_i) \in [\![\overline{Y}]\!]^{\mathcal{R},\alpha}$.

   Hence, $\mathcal{R}, \alpha \models \overline{X} \dot{\sqsubseteq} \forall f : \overline{Y}$.

2. if $x \dot{\sqsubseteq} X,\ X \dot{\sqsubseteq} f : \{\overline{X}_1, \ldots, \overline{X}_n\} \in C_s$ then we need to show that $\mathcal{R}, \alpha \models X \dot{\sqsubseteq} f : \{X_1, \ldots, X_n\}$.

   By definition of normal form we know that there exists $x = f : \{x_1, \ldots, x_m\} \in C_s$ such that $m \leq n$ and the following conditions hold:
   - for each $\overline{X}_i : 1 \leq i \leq n$ there exists $x_j : 1 \leq j \leq m$ such that either $x_j \dot{\sqsubseteq} \overline{X}_i \in C_s$ or $x_j \equiv \overline{X}_j$ and
   - for each $x_j : 1 \leq j \leq m$ there exists $\overline{X}_i : 1 \leq i \leq n$ such that either $x_j \dot{\sqsubseteq} \overline{X}_i \in C_s$ or $x_j \equiv \overline{X}_j$

   This means that $\alpha(X) \subseteq f : \{X_1, \ldots, X_n\}^R$.
   Hence, $\mathcal{R}, \alpha \models X \dot{\sqsubseteq} f : \{X_1, \ldots, X_n\}^R$.

3. if $x \dot{\sqsubseteq} X,\ X \dot{\sqsubseteq} \geq f : n \in C_s$ then we know that $|succ(x, f)| \geq n$.
   Hence, $\mathcal{R}, \alpha \models X \dot{\sqsubseteq} \geq f : n$.

Hence, we have the theorem.

3.8.3 Normalisation Rules for the language $\mathcal{S}_2$

In this section, we provide a set of normalisation rules for the language $\mathcal{S}_2$ that transforms any $\mathcal{S}_2$ constraint system into normal form. We shall demonstrate that our rules are invariant, complete and terminating thus obtaining a decision procedure for consistency of $\mathcal{S}_2$ constraint systems. Furthermore, since any $\mathcal{ALS}$ term T is consistent *iff* the $\mathcal{S}_2$ constraint system $\{x \doteq T\}$ is consistent, this means that we also have a decision procedure for the consistency of $\mathcal{ALS}$ terms.

The normalisation rules are grouped into 3 stages according to the classification scheme we employed for the language $\mathcal{L}_2$.

For the first stage of normalisation which pushes negations inwards, we augment the $\mathcal{L}_2$ normalisation rules with the following additional rule which eliminates negations of set descriptions.

Stage 1 : Rules for pushing negations inwards

1. $\{\overline{X} \doteq \neg\{T_1, \ldots, T_n\}\} \cup C_s$
   $\longrightarrow$
   $\{\overline{X} \doteq (\geq f : n+1 \sqcap \exists f : \top) \sqcup \forall f : \neg T_1 \sqcup \ldots \sqcup \forall f : \neg T_n \sqcup \exists f : (\neg T_1 \sqcap \ldots \sqcap \neg T_n)\}$

Once, the above rule in conjunction with the Stage 1 rules for the language $\mathcal{L}_2$ have been applied to an initial constraint system the resulting constraint system will contain only contain negations of atoms, constants or primitive concepts.

The second stage involves breaking down $\mathcal{ALS}$ terms into simpler ones by introducing new variables. For this purpose, we shall employ the results of theorem 3.8.1 which allows us to employ the following rules.

Stage 2 : Rules for breaking down $\overline{X} \doteq \Gamma$ constraints

1. $\{X \doteq f : \{T_1, \ldots, T_n\}\} \cup C_s \longrightarrow \{X \doteq f : \{X_1, \ldots, X_n\}, X_1 \doteq T_1, \ldots, X_n \doteq T_n\} \cup C_s$
   where each X_i $1 \leq i \leq n$ is new

2. $\{x \dot{\sqsubseteq} f : \{T_1, \ldots, T_n\}\} \cup C_s \longrightarrow \{x = f : \{x_1, \ldots, x_n\}, x_1 \dot{\sqsubseteq} T_1, \ldots, x_n \dot{\sqsubseteq} T_n\} \cup C_s$
   where each $x_i : 1 \leq i \leq n$ is new

Once, the above two Stages have been applied to a given constraint system, we are ready to apply the actual constraint propagation rules. The constraint propagation rules for $\mathcal{S}_2$ consists of all the normalisation rules for the language $\mathcal{S}_1$ and 3 additional rules presented below.

Notation: We shall write $x \dot{\sqsubseteq} \geq f : n$ to mean $x \dot{\sqsubseteq} \geq f : n : \emptyset$.

Stage 3 : Constraint Propagation rules

Group 0 : Additional rule for simplifying $\dot{\sqsubseteq}$ constraints

1. $\{x \dot{\sqsubseteq} f : \{\overline{X}_1, \ldots, \overline{X}_n\}\} \cup C_s \longrightarrow \{x = f : \{x_1, \ldots, x_n\}, x_1 \dot{\sqsubseteq} \overline{X}_1, \ldots, x_n \dot{\sqsubseteq} \overline{X}_n\} \cup C_s$
   where $x_1, \ldots, x_n$ are new.

Group 1 rules

All normalisation rules for the language $\mathcal{S}_1$

Additional Group 2 rules

2. $\{\overline{X} \dot{\sqsubseteq} \forall f : \overline{Y},\ x = f : \{x_1, \ldots, x_n\}\} \cup C_s$
   $\longrightarrow$
   $\{\overline{X} \dot{\sqsubseteq} \forall f : \overline{Y},\ x = f : \{x_1, \ldots, x_n\},\ x_1 \dot{\sqsubseteq} \overline{Y}, \ldots, x_n \dot{\sqsubseteq} \overline{Y}\} \cup C_s$
   if there exists $x_i : 1 \leq i \leq n$ such that $x_i \dot{\sqsubseteq} \overline{Y} \notin C_s$ and $x_i \not\equiv \overline{Y}$ and
   (a) $\overline{X} \equiv x$ or
   (b) $\overline{X} \equiv X$ and $x \dot{\sqsubseteq} X \in C_s$

3. $\{x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} f : \{\overline{X}_1, \ldots, \overline{X}_n\}\} \cup C_s \longrightarrow$
   $\{x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} f : \{\overline{X}_1, \ldots, \overline{X}_n\}, x = f : \{x_1, \ldots, x_n\}, x_1 \dot{\sqsubseteq} \overline{X}_1, \ldots, x_n \dot{\sqsubseteq} \overline{X}_n\} \cup C_s$
   if
   (a) each $x_i : 1 \leq i \leq n$ is *new* and
   (b) there is no $x = f : \{y_1, \ldots, y_m\} \in C_s$ where $m \leq n$ such that :

i. for each $\overline{X}_i : 1 \leq i \leq n$ there exists $y_j : 1 \leq j \leq m$ such that $y_j \dot{\sqsubseteq} \overline{X}_i \in C_s$ or $y_j \equiv \overline{X}_i$ and

ii. for each $y_j : 1 \leq j \leq m$ there exists $\overline{X}_i : 1 \leq i \leq n$ such that $y_j \dot{\sqsubseteq} \overline{X}_i \in C_s$ or $y_j \equiv \overline{X}_i$

4. $\{x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \geq f : n\} \cup C_s \longrightarrow \{x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \geq f : n, x \dot{\sqsubseteq} \geq f : n\} \cup C_s$
if there is no $x \dot{\sqsubseteq} \geq f : m : \phi \in C_s$ such that $m \geq n$

The above Group 2 rules are ordered in the order shown. Furthermore, we assume that rules 3 and rule 4 are applied last among all the Group 2 rules including the ones for the language $\mathcal{L}_2$. Rule 2 is applied immediately after the first $\mathcal{L}_2$ Group 2 rule **BDis** (for propagating disjunctions).

3.9 Invariance, Completeness and Termination

Proposition 3.9.1 [Invariance] *C_s is a consistent constraint system iff at least one instance of every applicable rule transforms C_s into a consistent constraint system.*

Theorem 3.9.2 [Completeness of Rewriting] *The application of the above normalisation rules to any given constraint system transforms the given constraint system into normal form.*

Proof: To prove the completeness claim, as for the language $\mathcal{L}_2$, we need to ensure that if a constraint system is not in normal form then one of the normalisation rules apply. For each of the additional conditions for normal form given in definition 3.8.4, it is easy to see that there is a corresponding rule. Hence, we have the theorem.

We shall not attempt to rigorously establish the termination claim since this turns out to be tedious exercise. Intuitively speaking, termination is obvious since from the point of view of termination each set description of the form $f : \{T_1, \ldots, T_n\}$ behaves as if it is a finite number of existential terms of the form $\exists f : T_1, \ldots, \exists f : T_n$.

Proposition 3.9.3 [Termination] *There is no infinite chain issuing from the application of the normalisation rules on any $\mathcal{S}_2$ constraint system.*

This concludes our study of the term language $\mathcal{ALS}$ which provides feature terms with *variables, set descriptions, existential* and *universal relation quantification* along

with all the propositional connectives. We have described an invariant, complete and terminating consistency procedure for determining the consistency of $\mathcal{ALS}$ terms. Our consistency checking procedure extends the consistency checking procedure that we developed for the language $\mathcal{ALV}$.

3.10 Issues on Extensionality

In this section, we shift our focus to the issue of extensionality that is traditionally associated with sets. *Extensionality* basically states that two sets having the same elements as members are equal. We shall describe what extensionality means within our framework and provide an extensional semantics for the term language $\mathcal{ALS}$ since our current semantics can be considered as *non-extensional* or *intensional.*

To illustrate this point more clearly, note that the constraint system

(28) $\quad C_s = \{x \neq y,\ x = f:\{x_1,\ldots,x_n\},\ y = f:\{x_1,\ldots,x_n\}\ \}$

is *satisfiable* under our semantics. But, within an **extensional** semantics such a system of constraints may not have a model *e.g.* in the familiar set theory if:

$$x = \{x_1,\ldots,x_n\},\ y = \{x_1,\ldots,x_n\}$$

then x and y are identical under extensionality. For this reason we shall label our semantics as **non-extensional.**

The questions we shall try to answer in this section is the following:

- *Is it possible to provide an alternative extensional semantics ?*
- *What are the consequences ?*

In trying to answer the first question, we shall first try to analyse why our semantics is intensional.

There are two reasons that makes our semantics *intensional.*

The first one is inherent to feature logics. For instance, in the following:

(29) $C_{s1} = \{x = f : \{x\}, y = f : \{y\}, x \neq y\}$

the constraints of the variables x and y are identical modulo variable renamings. However this does not imply that they have the same model in every interpretation. The variables x and y can be interpreted differently simply by assuming that for some variable assignment α and a feature g distinct from f: $g^{\mathcal{I}}(\alpha(x)) \downarrow$ and $g^{\mathcal{I}}(\alpha(y)) \uparrow$

The second reason our semantics is *intensional* is because we have provided an *open-world* semantics. In other words, the semantic structure $< \mathcal{U}^I, .^I >$ is arbitrary with the only requirement that it behaves as a relational algebra. Since we know that the *relational graph algebra* $\mathcal{R}$ provides canonical interpretations, to illustrate this point it is enough that we restrict our attention to the relational graph algebra.

In the relational graph algebra we know that within the universe $\mathcal{U}^R$ the nodes of every relational graph are drawn from $\mathcal{V} \cup \mathcal{A}t$. This means that every variable x is distinct from every other variable y. Following the construction provided by the Normal form theorem for the language $\mathcal{L}_1$, a canonical interpretation for the above constraint system C_{s1} can be given simply by:

- $\alpha(x) = (x, \{xfx\})$
- $\alpha(y) = (y, \{yfy\})$

Note that the graphs $(x, \{xfx\})$ and $(y, \{yfy\})$ are distinct purely because x and y are distinct.

The above discussion outlines two different causes why two distinct variables can be interpreted distinctly within the interpretation structure provided by relational algebras. This means that if our aim is to provide an extensional semantics then we must block the above two mechanisms.

To counteract the first cause, we may restrict our alphabet to $\mathcal{F} = \mathcal{F}s \cup \{\eta\}$ where every symbol $f \in \mathcal{F}s$ is interpreted as a feature symbol *i.e.* every feature symbol is interpreted as a unary partial function $f^I : \mathcal{U}^I \rightharpoonup \mathcal{U}^I$ and the distinguished symbol η is interpreted as a binary relation *i.e.* $\eta^I \subseteq \mathcal{U}^I \times \mathcal{U}^I$. The relation η^I is intended to

model the set-membership relation $\in$.

This gives rise to the following new definition of a relational algebra.

Definition 3.10.1 [Extensional Relational Algebra] *A extensional relational algebra is a structure $\mathcal{I} = < \mathcal{U}^I, .^I >$ such that:*

1. *$\mathcal{U}^I$ is an arbitrary non-empty set.*
2. *$.^I$ is an interpretation function that interprets:*
   - *every symbol in $\mathcal{F}s$ as a unary partial function $f^I : \mathcal{U}^I \rightharpoonup \mathcal{U}^I$*
   - *η is interpreted as a binary function $\eta^I \subseteq \mathcal{U}^I \times \mathcal{U}^I$.*
3. *Furthermore, we shall require that every relational algebra $\mathcal{I}$ satisfy the following condition:*
   - *for atoms a_1, a_2, if $a_1 \not\equiv a_2$ then $a_1^I \neq a_2^I$*
   - *for any atom $a \in \mathcal{A}t$ and for any relation $f \in \mathcal{F}$: $f^I(a^I)\uparrow$*
   - *for every $e \in \mathcal{U}^I$ if $\eta(e)\downarrow$ then for every feature $f \in \mathcal{F}s$: $f^I(e)\uparrow$*
   - *conversely if for any feature $f \in \mathcal{F}s$, $f^I(e)\downarrow$ implies $\eta(e)\uparrow$*

Extensional relational algebras alone however is insufficient to provide an extensional semantics. But firstly we shall explain what this definition provides.

From the above definition of extensional relational algebras, η is the only available relational symbol. Thus instead of writing:

(30) $\quad C_s = \{x \neq y,\ x = \{x_1, \ldots, x_n\},\ y = \{x_1, \ldots, x_n\}\}$

we can add η to write:

(31) $\quad C_s = \{x \neq y,\ x = \eta : \{x_1, \ldots, x_n\},\ y = \eta : \{x_1, \ldots, x_n\}\}$

Now it follows that constraint systems such as:

(32) $\quad C_s = \{x = \eta : \{x\},\ y = \eta : \{y\}, x \neq y\}$

have the following model in the extensional relational graph algebra:

- $\alpha(x) = (x, \{x \eta x\})$

- $\alpha(y) = (y, \{y\eta y\})$

Note that x and y are still distinct simply because the extensional relational algebra would not require that x are y to be identical.

We now need to counteract the second cause. We need to somehow allow only relational algebras such that graphs such as $(x, \{x\eta x\})$ and $(y, \{y\eta y\})$ are treated as identical.

We shall demonstrate that any extensional relational algebra can be tranformed into relational algebras such that all objects that are extensionally identical belong to the same equivalence class. In other words, objects in the transformed relational algebra are equivalence classes of objects modulo extensionality.

$$C_s = \{x \neq y,\ x = \eta : \{x_1, x_2, x_3\},\ y = \eta : \{x_1, x_2, x_3\},\ x_1 : a, x_2 : b, x_3 : c\}$$

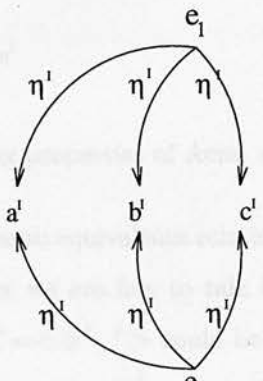
$\mathcal{U}^I = \{e_1, e_2, a^I, b^I, c^I\}$

$| \mathcal{U}^I | = 5$

$\eta^I = \{(e_1, a^I), (e_1, b^I), (e_1, c^I), (e_2, a^I), (e_2, b^I), (e_2, c^I)\}$

$\alpha(x) = e_1$

$\alpha(y) = e_2$



$\mathcal{U}^I / \approx^I = \{[e_1], [a^I], [b^I], [c^I]\}$

$| \mathcal{U}^I / \approx^I | = 4$

$[e_1] = [e_2] = \{e_1, e_2\}$

$[a^I] = \{a^I\}$

$[b^I] = \{b^I\}$

$[c^I] = \{c^I\}$

Figure 3.7: Intensional *vs.* Extensional Semantics

The notion of extensionality we shall adopt is due to [Aczel 88] since it provides a notion of extensionality for sets containing circularities or non-well-founded sets. Note that the sets in our relational algebras are potentially non-well-founded.

An extensional semantics identifies certain elements in the domain $\mathcal{U}^I$ as belonging to the same equivalence class that is generated by a *bisimulation relation* defined as follows:

Definition 3.10.2 [Bisimulation] *We say that two elements e_1 and e_2 in $\mathcal{U}^I$ are bisimilar (written $e_1 \approx^I e_2$) iff there exists a relation R s.t. :*

1. $e_1 R e_2$ *if* $e_1 = e_2$

2. *For each feature* f^I:
   $f^I(e_1)\downarrow \Rightarrow f^I(e_2)\downarrow \wedge f^I(e_1)Rf^I(e_2)$ *and conversely*
   $f^I(e_2)\downarrow \Rightarrow f^I(e_1)\downarrow \wedge f^I(e_1)Rf^I(e_2)$
3. *For each* $e_r \in \mathcal{U}^I$ *s.t.* $e_1\ \eta^I\ e_r$ *there exists* $e_s \in \mathcal{U}^I$ *s.t.* $e_2\ \eta^I\ e_s$ *and* e_rRe_s *and*
4. *For each* $e_s \in \mathcal{U}^I$ *s.t.* $e_2\ \eta^I\ e_s$ *there exists* $e_r \in \mathcal{U}^I$ *s.t.* $e_1\ \eta^I\ e_r$ *and* e_rRe_s

This definition of bisimulation is essentially the same as given by [Rounds 88]. The last two items in the above definition are as given by [Aczel 88] and the rest have been adopted to deal with features and atoms.

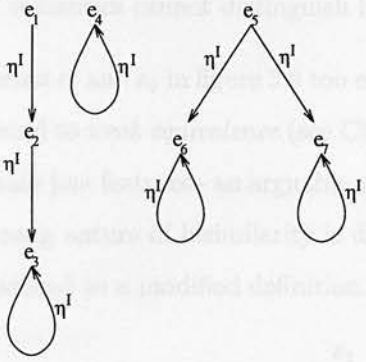


Figure 3.8: Strong properties of Aczel style bisimilarity

The notion of bisimulation defines an equivalence relation on the elements of $\mathcal{U}^I$ relative to an interpretation $\mathcal{I}$ and hence we are free to talk in terms of equivalence classes. Thus, any given interpretation $\mathcal{I} = <\mathcal{U}^I, .^I>$ could be restricted to the interpretation $\mathcal{I} = <\mathcal{U}^I/\approx^I, .^I>$ where each element in $\mathcal{U}^I$ is an equivalence class of elements. Each function f^I and the relation η^I will appropriately need to be extended to deal with equivalence classes. We refer to this semantics as the *extensional semantics*. This answers the first question in a positive way.

The above discussion should make it clear that to provide an extensional semantics all that is needed is an equivalence relation on $\mathcal{U}^I$. Thus while bisimulation defines an equivalence relation, yet another equivalence relation we could employ would be the notion of *weak equivalence* that we defined in Chapter 2[2].

[2]Drew Moshier has suggested to me (personal communication) that yet another notion of *weak bisimulation* relation that one could adopt is that which requires identity on "functional" nodes and bisimulation on "set" nodes.

To illustrate the effects of an extensional semantics, we shall look at some examples.

In the example in figure 3.7 the elements e_1 and e_2 are bisimilar. Furthermore, for any $\mathcal{I}$-interpretation $\alpha(x) \approx^I \alpha(y)$ and hence the constraint system C_s in figure 3.7 is unsatisfiable according to the extensional semantics while it is satisfiable under the intensional semantics.

It turns out that Aczel's notion of extensionality can often be too strong. For instance, according to our definition, every pair of elements $e_1, e_2, \ldots, e_7$ in figure 3.8 are bisimilar and hence our extensional semantics cannot distinguish between them.

Furthermore, the two elements e_1 and e_2 in figure 3.9 too are bisimilar. This means that bisimilarity cannot be reduced to *weak equivalence* (see Chapter 2 for a definition) if we restrict graph edges to contain just features - an arguably undesirable consequence. The principal reason for the strong nature of bisimilarity is due to its relational definition which clearly cannot be reduced to a modified definition of partial endomorphism.

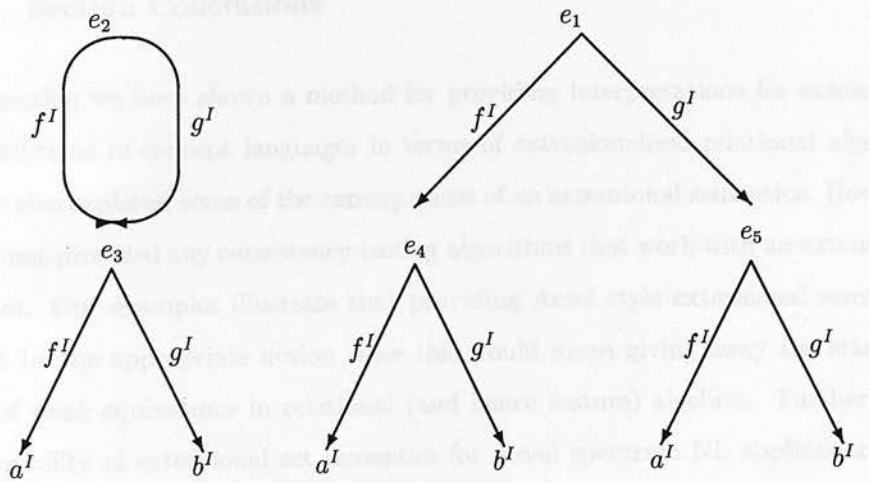


Figure 3.9: e_1 and e_2 are Aczel bisimilar

In terms of satisfiability of constraint systems this means that constraint systems such as

$$C_{s1} = \{x \neq y,\ x = \eta : \{x\},\ y = \eta : \{z\},\ z = \eta : \{z\}\}$$

and

$$C_{s2} = \{x \neq y,\ x = \eta : \{x\},\ y = \eta : \{z, w\},\ z = \eta : \{z\},\ w = \eta : \{w\}\}$$

are *unsatisfiable* under the extensional semantics.

In the above discussion we have motivated the issues using constraint systems from the language $\mathcal{S}_1$. The question then arises as to what results carry over to the term language $\mathcal{ALS}$.

From the semantics of $\mathcal{ALS}$ terms we know that for any term T: $T \sqcap \neg T \equiv \bot$. It follows that for any set description $f : \{T_1, \ldots, T_n\} \sqcap \neg f : \{T_1, \ldots, T_n\} \equiv \bot$.

However the term $f : \{f : \{x\}\}$ is not equivalent to $f : \{x\}$ and this means that $f : \{f : \{x\}\} \sqcap f : \{x\} \not\equiv \bot$.

Hence, $\mathcal{ALS}$ can be considered to exhibit only a *weak* form of extensionality (in the Aczel sense).

3.10.1 Section Conclusions

In this section we have shown a method for providing interpretations for extensional set descriptions in concept languages in terms of extensionalised relational algebras. We have also explored some of the consequences of an extensional semantics. However, we have not provided any consistency testing algorithms that work with an extensional semantics. Our examples illustrate that providing Aczel style extensional semantics may not be the appropriate notion since this would mean giving away the standard notion of *weak equivalence* in relational (and hence feature) algebras. Furthermore, the desirability of extensional set semantics for broad spectrum NL applications has yet to be demonstrated.

Hence, we conclude that providing strong extensional semantics in a language such as $\mathcal{ALS}$ raises more questions than it solves.

3.11 Summary

In this chapter we have provided a method for incorporating so called partial descriptions of sets into concept languages. Our approach provides a precise meaning for set descriptions. We have shown that set descriptions have a close analog to what are known as number restrictions in concept languages. However, our translation shows that it is more efficient to keep set descriptions than to eliminate them in favour of number restrictions. Our translation also makes clear what negative set descriptions mean - something that has not been considered in computational linguistics. Our normalisation rules provide a method for consistency testing of concept terms containing set descriptions. This is again something that had not been considered previously in computational linguistics. We finally show what incorporating extensionality with set descriptions in a concept language would mean. This highlights some potential problems that need to be considered if extensionality is to be considered seriously.

Chapter 4

Embedding Order Relations in Constraint Languages

4.1 Introduction

Current constraint based grammars often assume the availability of an intuitively understood mechanism to express *linear precedence* constraints to deal with the phenomena of *word-order* in natural language. It is assumed that constraint based approaches to word-order provide greater flexibility and representational economy to express word-order constraints as compared to ID/LP based mechanisms (see for instance [Uszkoreit 85], [Pollard & Sag 92]). However, there appears to be hardly any work on appropriate constraint logics to deal with linear precedence constraints although there have been various formalisations of linear precedence constraints - [Uszkoreit 85] provides a modification to the GPSG ID/LP format to allow LP constraints that are *ordered* and sensitive to other grammatical information; [Reape 89] provides a denotational account of linear precedence constraints on so called *word-ordering domains* that can be thought of as partially ordered strings.

In this Chapter we provide a concrete formalisation of a constraint logic that accommodates linear precedence constraints as a specialisation of *strict order relations*. We shall extend both the constraint language $\mathcal{S}_1$ and develop a term language with constraints for expressing order.

4.2 Motivation

In this section, we shall look at some representative natural language examples involving word-order. These examples will lead us to our choice of constraints for expressing order.

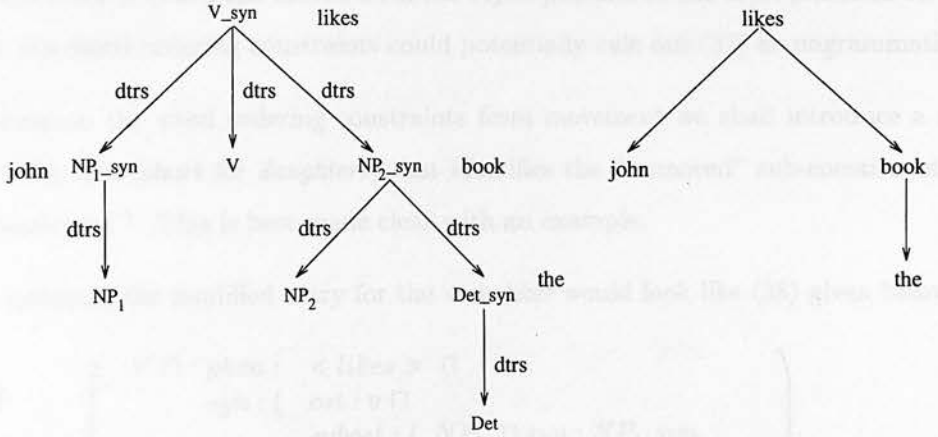


Figure 4.10: The *dtrs* attribute and associated dependency diagram

Consider the following simplified lexical entry for the verb *likes*.

$$(33) \qquad \left(\begin{array}{ll} V \sqcap & phon: \ < likes > \sqcap \\ & syn: (\ cat: v \sqcap \\ & \quad subcat: \{NP_1, NP_2, \}) \sqcap \\ NP_1 \prec\prec V \sqcap & \\ V \prec\prec NP_2 & \end{array} \right)$$

The constraint $NP_1 \prec\prec V$, $V \prec\prec NP_2$ is intended to capture the fact that in declarative English sentences, the common word order is *SVO* (*i.e. subject, verb, object*). Thus, the above lexical entry is intended to licence the grammaticality of sentences such as:

(34) John likes Mary.

(35) That man likes Mary.

(36) I like the book on the table.

We assume that grammatical constraints other than that specified in (33) will identify the correct *subject* and *object* with NP_1 and NP_2 respectively.

However, it is not clear how the above specification works when the phenomenon of *movement* occurs. For instance, in examples such as:

(37) Mary$_i$, John likes $\_i$.

the constituent *Mary* has moved from the *object* position to the *topic* position. In this case, our word ordering constraints could potentially rule out (37) as ungrammatical.

To insulate the word ordering constraints from movement we shall introduce a new attribute *dtrs* (short for *daughters*) that identifies the "unmoved" sub-constituents of a constituent [1]. This is best made clear with an example.

For instance, the modified entry for the verb *likes* would look like (38) given below.

(38)
$$\left(\begin{array}{l} V \sqcap \; phon: \; < likes > \sqcap \\ \qquad syn: (\; cat: v \sqcap \\ \qquad\qquad subcat: \{\; NP_1 \sqcap syn: NP_1\_syn, \\ \qquad\qquad\qquad\qquad NP_2 \sqcap syn: NP_2\_syn\} \sqcap \\ \qquad\qquad dtrs: \{V, NP_1\_syn, NP_2\_syn\}) \sqcap \\ NP_1\_syn \prec\prec V \sqcap \\ V \prec\prec NP_2\_syn \end{array}\right)$$

(39)
$$\left(\begin{array}{l} NP_1 \sqcap \; phon: \; < john > \sqcap \\ \qquad syn: (\; cat: n \sqcap \\ \qquad\qquad subcat: \{\} \sqcap \\ \qquad\qquad dtrs: \{NP_1\}) \end{array}\right)$$

(40)
$$\left(\begin{array}{l} NP_2 \sqcap \; phon: \; < book > \sqcap \\ \qquad syn: (\; cat: n \sqcap \\ \qquad\qquad subcat: \{DET \sqcap syn: DET\_syn\} \sqcap \\ \qquad\qquad dtrs: \{NP_2, DET\_syn\}) \\ DET\_syn \prec\prec NP_2 \end{array}\right)$$

(41)
$$\left(\begin{array}{l} DET \sqcap \; phon: \; < the > \sqcap \\ \qquad syn: (\; cat: det \sqcap \\ \qquad\qquad subcat: \{\} \sqcap \\ \qquad\qquad dtrs: \{DET\}) \end{array}\right)$$

[1] Our usage of the *dtrs* attribute is similar to the usage of the *dtrs|local* attribute in HPSG [Pollard & Sag 87].

The idea is that the value of the *syn:dtrs* attribute are all the "unmoved" constituents. The constraint $V \prec\prec NP_2\_syn$ in (38) is then intuitively meant to enforce the constraint that the variable V should precede every "node" reachable through the attribute *dtrs* of the variable NP_2_syn.

In the case of declarative sentences such as in (34) the values of the *dtrs* attribute will be as shown in figure 4.10. The variable V_syn in figure 4.10 is intended to stand for the value of the path *syn* for the variable V in example (38) and similarly for other phrases.

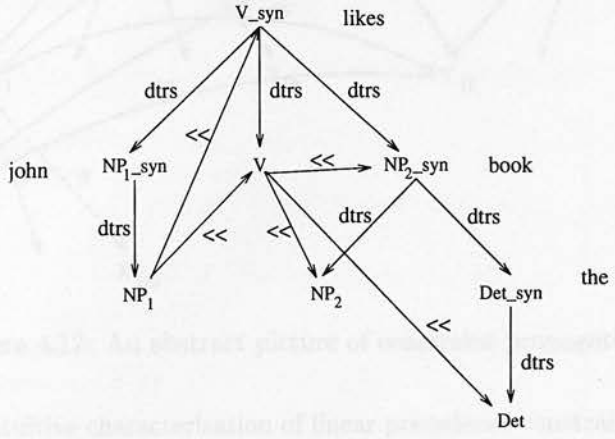


Figure 4.11: A partial propagation of *linear precedence* constraints

This means that, in a phrase such as [*john*] *likes* [*the book*] our ordering constraints given in (38) should imply the ordering constraints as depicted in fig 4.11. This shows that the constraint $V \prec\prec NP_2\_syn$ should somehow imply both *likes* $\prec\prec$ *the* and *likes* $\prec\prec$ *book*.

On the other hand, in a topicalised sentence such as:

(42) The book, John likes —.

we shall assume that grammatical principles will assign the value of the *dtrs* attribute of NP_2 to be $\emptyset$ and hence the word ordering constraints specified in (38) will vacuously be satisfied. As such this will not prevent the verb coming before its object. To prevent this an analog of the *slash* feature would be needed and a corresponding LP constraint which forces both the *subject* and the *verb* to come after the "slashed" item.

In a more abstract setting, given a *distinguished* relation symbol p, the constraint $x \prec\prec y$ should entail that for every element x_p which is a p-value of x and for every element y_p which a p-value of y, $x_p \prec\prec y_p$ should hold (see fig 4.12). In the above and subsequent examples in this section we assume that the distinguished relation symbol p is just the relational attribute *dtrs*.

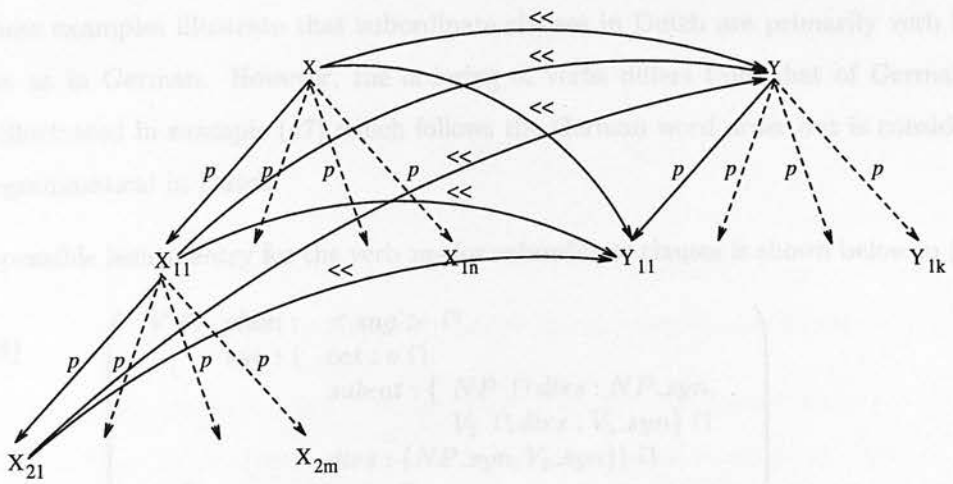


Figure 4.12: An abstract picture of constraint propagation

With the above intuitive characterisation of linear precedence constraints we shall look at some more motivating examples before we proceed to develop a formal characterisation.

Consider the following examples adapted from [Steedman 85] which are intended to reflect the word order of Dutch *subordinate clauses.*

(43) omdat ik$_1$ Cecilia$_2$ de nijlpaarden$_2$ zag$_1$ voeren$_2$
because I Cecilia the hippos saw feed
because I saw Cecilia feed the hippos

(44) omdat ik$_1$ Cecilia$_2$ Henk$_3$ de nijlpaarden$_3$ zag$_1$ helpen$_2$ voeren$_3$
because I Cecilia Henk the hippos saw help feed
because I saw Cecilia help Henk feed the hippos

(45) omdat ik Cecilia Henk de nijlpaarden zag helpen voeren

(46) *omdat ik Cecilia zag Henk de nijlpaarden helpen voeren

(47) *omdat ik Cecilia Henk de nijlpaarden voeren helpen zag

Identical numbers in a nominal argument and a verb in the above examples is intended to reflect the fact that the nominal argument is an argument of the verb with the same number.

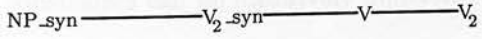
These examples illustrate that subordinate clauses in Dutch are primarily verb final just as in German. However, the ordering of verbs differs from that of German as is illustrated in example (47) which follows the German word order but is considered ungrammatical in Dutch.

A possible lexical entry for the verb *zag* for subordinate clauses is shown below in (48).

(48)
$$\left(\begin{array}{l} V \sqcap \; phon: \; < zag > \sqcap \\ \qquad syn: (\; cat: v \sqcap \\ \qquad\qquad subcat: \{ \; NP \sqcap dtrs: NP\_syn, \\ \qquad\qquad\qquad\qquad V_2 \sqcap dtrs: V_2\_syn\} \sqcap \\ \qquad\qquad dtrs: \{NP\_syn, V_2\_syn\}) \sqcap \\ NP\_syn \prec\prec V_2\_syn \sqcap \\ NP\_syn \prec\prec V \sqcap \\ V_2\_syn \prec\prec V \sqcap \\ V \prec\prec V_2 \end{array}\right)$$

In this example, note that the *syn* : *dtrs* attribute of the verb does not contain the verb itself in contrast to the English case. This choice results in a much easier representation of the required ordering constraints. However, the word ordering constraints specify the relative position of the main verb with respect to the subcategorised verb.

The ordering constraints enforced by the above example can be intuitively understood to be as given in (49).

(49)


The first word ordering constraint $NP\_syn \prec\prec V_2\_syn$ in (48) state that the *subject NP* of *zag* (- *ik* in the example in figure 4.13) precedes all the *nominal* arguments of V_2 (- *Cecilia Henk de nijlpaarden* in the example in figure 4.13). This in turn precedes *zag* according to the third word ordering constraint $V_2\_syn \prec\prec V$. The final word ordering constraint $V \prec\prec V_2$ states that *zag* precedes the embedded verb V_2 (- *helpen*

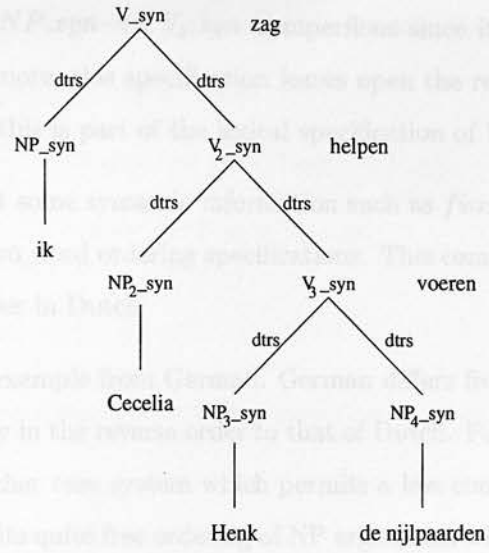


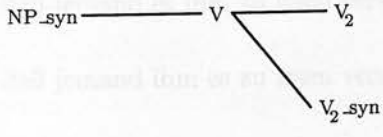
Figure 4.13: The *dtrs* attribute for Dutch subordinate clauses

in figure 4.13). The constraint $NP\_syn \prec\prec V$ is superfluous since it follows from the other constraints. However, we have added this constraint in order to highlight the way in which the word ordering in the main clause in Dutch contrasts with that in the subordinate clause.

Dutch main clauses follow the *verb-second* phenomenon in that the finite verb is realised usually as the word immediately after the first NP in the main clause. For these cases we can use the word-ordering constraints given below in (50).

(50)
$$NP\_syn \prec\prec V_2\_syn \sqcap$$
$$NP\_syn \prec\prec V \sqcap$$
$$V \prec\prec V_2\_syn \sqcap$$
$$V \prec\prec V_2$$

This set of constraints can be intuitively understood to enforce the following ordering constraints.

(51)


The only difference in (50) from the entry given in (48) is that the word ordering constraint $V_2\_syn \prec\prec V$ is replaced by the constraint $V \prec\prec V_2\_syn$. Note that in this

case, the constraint $NP\_syn \prec\prec V_2\_syn$ is superflous since it follows from the other constraints. Furthermore, this specification leaves open the relative ordering between V_2 and V_2_syn since this is part of the lexical specification of V_2 (as it should be).

We shall assume that some syntactic information such as $fin\pm$ will trigger the choice between the above two word ordering specifications. This completes our brief example of handling word order in Dutch.

We next look at an example from German. German differs from Dutch in that clause final verbs are usually in the reverse order to that of Dutch. Furthermore, German has a morphologically richer case system which permits a less constrained word ordering. In particular it permits quite free ordering of NP arguments within a single verb group [Uszkoreit 85] and secondly it permits the so called *scrambling* or re-ordering of NP arguments within multiple verb groups (see for instance [Reape 93]). However, like Dutch, German also exibits the V_2 phenomenon.

Consider the following examples from [Reape 93] which are intended to reflect the word order of German *subordinate clauses.*

(52) daß es_3 ihm_2 $jemand_2$ zu $lesen_3$ $versprochen_2$ hat_1.
that it[ACC] him[DAT] someone[NOM] to read promised has.
that someone has promised him to read it.

(53) daß es ihm jemand zu lesen versprochen hat.

(54) daß es jemand ihm zu lesen versprochen hat.

(55) daß ihm es jemand zu lesen versprochen hat.

(56) daß ihm jemand es zu lesen versprochen hat.

(57) daß jemand es ihm zu lesen versprochen hat.

(58) daß jemand ihm es zu lesen versprochen hat.

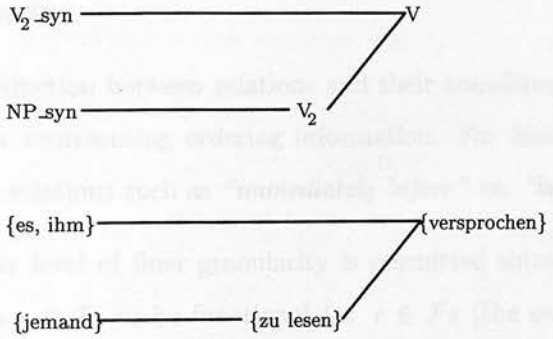
The above examples illustrate that in constrast to Dutch German permits free permutations of NP arguments (*i.e.* scrambling).

The following is a plausible entry for the verb *versprechen* in subordinate clauses [2].

(59)
$$\left(\begin{array}{ll} V \sqcap & phon: \ < versprochen > \sqcap \\ & syn: (\ cat: v \sqcap \\ & \quad subcat: \{ \ NP \sqcap syn: NP\_syn, \\ & \qquad\qquad V_2 \sqcap syn: V_2\_syn\} \sqcap \\ & \quad dtrs: \{NP\_syn, V_2\_syn\}) \sqcap \\ V_2\_syn \prec\prec V \sqcap & \\ NP\_syn \prec\prec V_2 \sqcap & \\ V_2 \prec\prec V & \end{array}\right)$$

Intuitively, the above specification can be understood to be enforcing the constraints as depicted in (60).

(60)



This entry lacks a specification for the relative ordering between NP_syn and V_2_syn and the ordering between these is free. Furthermore, the above specification is also agnostic about the relative ordering between V_2_syn and V_2 and we assume that the specification of V_2 will correctly specify the right ordering.

Assuming that the specification of the other verbs are appropriately specified, this specification would admit all the examples (53) through (58).

4.3 The syntax and semantics of order relations

In this section, we develop a theory of linear precedence constraints as constraints on order relations. For this purpose, we extend the syntax and semantics of relation symbols as follows.

[2] We disregard any considerations of the phenomena of *control* here

Let $\mathcal{F}i \subseteq F$ be a subset of *strict order relation* symbols. For every relational algebra $\mathcal{I}$ and every *strict order symbol* $r \in \mathcal{F}i$:

$$\text{if } (x, y) \in (r^I)^+ \text{ then } x \neq y$$

where $(r^I)^+$ denotes the transitive closure of the relation r^I.

Our definition of order relations is slightly different from the ones that are conventionally assumed (for instance see [Partee *et al* 90] *pp.* 47) in that every $r \in \mathcal{F}i$ is not necessarily a transitively closed strict order relation but its transitive closure is a strict order relation. Thus members of $\mathcal{F}i$ are *irreflexive* relations that define the skeleton of a strict order.

This distinction between relations and their transitive closure provides a finer granularity for representing ordering information. For instance, it allow us to distinguish between relations such as *"immediately before" vs. "before"*.

A further level of finer granularity is permitted since we are free to designate some elements $r \in \mathcal{F}i$ to be functional *i.e.* $r \in \mathcal{F}s$ (the set of feature symbols). In other words, the condition $\mathcal{F}i \cap \mathcal{F}s = \emptyset$ is **not** imposed by our definition. This will permit us to specify both *linear orders* as well as *branching orders.*

The syntax of relation symbols is extended to permit the specification of *transitive closure* of order relation symbols by allowing relation symbols of the form r^+ where $r \in \mathcal{F}i$ is a strict order relation symbol.

The interpretation of r^+ is provided in an obvious way by the following definition:

$$(r^+)^I = (r^I)^+ \quad \text{where } r \in \mathcal{F}i$$

Our aim now is to extend the constraint solving machinery of the language $\mathcal{S}_1$ to allow constraints involving strict order relations. For this purpose, we introduce the following additional constraints:

$$x\ r^+\ y, \quad p^*(x) \times q^*(y) \doteq r, \quad p^*(x) \times q^*(y) \doteq r^+$$

where $r \in \mathcal{F}i$ is a strict order relation symbol; $p, q \in \mathcal{F}$ are relation symbols and $x, y \in \mathcal{V}$ are variables.

Our syntax rules out *negative* ordering constraints such as $p^*(x) \times q^*(y) \not\dot{\sqsubseteq} r^+$. Furthermore, constraints of the form $x\ r^+\ y$ are only permitted if r is a strict order relation.

Given a relational algebra $\mathcal{I}$ and an $\mathcal{I}$-assignment α, the interpretation of each of these additional constraints is provided by the following definitions:

1. $\mathcal{I}, \alpha \models x\ r^+\ y \iff (\alpha(x), \alpha(y)) \in (r^I)^+$
2. $\mathcal{I}, \alpha \models p^*(x) \times q^*(y) \dot{\sqsubseteq} F \iff (p^*)^I(\alpha(x)) \times (q^*)^I(\alpha(y)) \subseteq F^I$

where we have abbreviated F to range over r, r^+.

Note that according to the above definition, $x \times y \dot{\sqsubseteq} r \equiv x\ \exists r\ y$ and $x \times y \dot{\sqsubseteq} r^+ \equiv x\ r^+\ y$, thus the forms $x\ \exists r\ y$ and $x\ r^+\ y$ are redundant.

With respect to the above constraints, the constraint of the form $x \prec\!\prec y$ (where x, y are variables) as used in section 4.2 can be understood simply a shorthand for the constraint $dtrs^*(x) \times dtrs^*(y) \dot{\sqsubseteq} \prec^+$, where $\prec \in \mathcal{F}i \cap \mathcal{F}s$ is a strict functional relation that relates two adjacent lexical items.

The language $\mathcal{T}_1$ is the language $\mathcal{S}_1$ plus the above constraints.

4.4 Normalisation of $\mathcal{T}_1$ constraint systems

In this section we consider the determination of consistency of $\mathcal{T}_1$ constraint systems. Our first step is to define a normal form from which it becomes easy to determine whether a given constraint system is consistent or inconsistent.

4.4.1 Normal Form

First some definitions.

Definition 4.4.1 *We say that* $x\ \dot{f}\ x_i \in C_s$ *where* f *is a relation symbol if:*

- *either* $x f x_i \in C_s$

- *or* $x\ \exists f\ x_i \in C_s$
- *or* $x = f : \{x_1, \ldots, x_i, \ldots, x_n\} \in C_s$

Similarly, let $x\ \dot{f}^+\ y \in C_s$ *if either* $x\ \dot{f}\ y \in C_s$ *or* $x\ f^+\ y \in C_s$

Let $x\ \dot{f}^*\ y \in C_s$ *if either* $y \equiv x$ *or* $x\ \dot{f}^+\ y \in C_s$.

We are now ready to present the normal form for the language $\mathcal{T}_1$.

Definition 4.4.2 [Normal Form] *We say that a* $\mathcal{T}_1$ *constraint system* C_s *is in normal form if* C_s *satisfies the following conditions:*

1. C_s *satisfies all the conditions for normal form for the language* S_1.
2. *if* $xry \in C_s$ *then* $x\ r^+\ y \notin C_s$.
3. *if* $x\ \exists r\ y \in C_s$ *then* $x\ r^+\ y \notin C_s$.
4. *if* $p^*(x) \times q^*(y) \dot{\sqsubseteq} r \in C_s$ *then* $p^*(x) \times q^*(y) \dot{\sqsubseteq} r^+ \notin C_s$
5. *if* $x\ \dot{r}^+\ y \in C_s$ *and* $y\ \dot{r}^+\ z \in C_s$ *then* $x\ \dot{r}^+\ z \in C_s$
6. *if* $p^*(x) \times q^*(y) \dot{\sqsubseteq} F \in C_s$ *then for each* $x\ \dot{p}^*\ x_i \in C_s$ *and each* $y\ \dot{q}^*\ y_i \in C_s$: $p^*(x_i) \times q^*(y_i) \dot{\sqsubseteq} F' \in C_s$ *such that the following dependency hold between* F *and* F':
   - *if* $F \equiv r$ *then* $F' \equiv r$
   - *if* $F \equiv r^+$ *then* $F' \in \{r, r^+\}$

Note that with respect to condition 6, if $p^*(x) \times q^*(y) \dot{\sqsubseteq} r^+ \in C_s$ then it is not strictly necessary to have $x\ r^+\ y \in C_s$ since in this case if either $x\ \exists r\ y \in C_s$ or $xry \in C_s$ then any interpretation that satisfies $x\ \exists r\ y$ or xry will also satisfy $x\ r^+\ y$.

We say that a constraint system C_s in normal form contains a **clash** if *either* of the following conditions hold:

1. if C_s contains a clash according to the definition of clash for $\mathcal{S}_1$ constraint systems.
2. if $xFx \in C_s$ where F ranges over $r, \exists r, r^+$ such that $r \in \mathcal{F}i$.

Theorem 4.4.3 [Normal Form] *The normal form theorem for* $\mathcal{S}_1$ *constraint systems can be extended to provide solutions to every clash-free* $\mathcal{T}_1$ *constraint system.*

Proof: It is obvious that if a $\mathcal{T}_1$ constraint system C_s contains a clash then there is no interpretation that satisfies C_s.

Let $\mathcal{R}, \alpha$ be the interpretation constructed by the normal form theorem for the language $\mathcal{S}_1$.

To show that every $\mathcal{T}_1$ constraint system in normal form not containing a *clash* is consistent we need to show that every clash-free $\mathcal{T}_1$ constraint system C_s is *satisfied* by the interpretetation $\mathcal{R}, \alpha$ constructed by the normal form.

In order to deal correctly with transitive constraints of the form $x\ r^+\ y \in C_s$, we need to make the following minor addition to the way relation symbols are interpreted by the normal form theorem for $\mathcal{S}_1$.

> If $xr^+y \in C_s$ such that there is **no** sequence of constraints:
> $x\ \dot{r}\ x_1, x_1\ \dot{r}\ x_2, \ldots, x_n\ \dot{r}\ y \in C_s$
> then in this case it is clear that the transitive closure of r^R will not contain $(\alpha(x), \alpha(y))$.
>
> We extend the interpretation of r^R in such as way that it does not affect the interpretation of other existing constraints as follows.
>
> Add $(\alpha(x_n), \alpha(y)) \in r^I$ where $xrx_1, x_1rx_2, \ldots, x_{n-1}rx_n \in C_s$ where $n \geq 0$ such that there is no constraint $x_nrz \in C_s$ for some variable z. Note that there can be no cycles of the form $xrx_1, x_1rx_2, \ldots, x_{n-1}rx \in C_s$ since this would mean that either $xrx \in C_s$,$x\ \exists r\ x \in C_s$ or $x\ r^+\ x \in C_s$ (from condition 5) and hence would result in a *clash*.
>
> Thus if $xrx_1, x_1rx_2, x_2rx_3, xr^+y \in C_s$ but there is no z such that $x_3rz \in C_s$, then the above interpretation effectively adds $x_3\ \exists r\ y$ into C_s. From this it follows that $(\alpha(x_3), \alpha(y)) \in r^R$ and hence $(\alpha(x_3), \alpha(y)) \in (r^R)^+$. Also $x \not\equiv z$ and so we cannot be making the relation $(r^R)^+$ illegal.

The above construction does not contradict the interpretation of r^R since the language $\mathcal{T}_1$ does not permit constraints of the form $x_3r\uparrow$, $x_3\ \forall r\ z$ or negative constraints of the form $\neg x\ \exists r\ z$. Hence, assuming $(\alpha(x_3), \alpha(y)) \in (r^R)^+$ cannot contradict the existing

constraints.

Definition 4.4.4 *Let* $\vec{f}^{+}$ *denote the transitive closure of the relation* $\dot{f}^{+}$ *given as the least relation satisfying:*

- *if* $x \ \dot{f}^{+} \ y \in C_s$ *then* $x \ \vec{f}^{+} \ y$
- *if* $x \ \vec{f}^{+} \ y$ *and* $y \ \vec{f}^{+} \ z$ *then* $x \ \vec{f}^{+} \ z$

The idea with the $\vec{f}^{+}$ relation is that given the constraint $p^*(x) \times q^*(y) \dot{\sqsubseteq} F$ where F ranges over r, r^+ our normal form ensures that every pair of variables x', y' such that $x \ \vec{p}^{+} \ x'$ and $y \ \vec{q}^{+} \ y'$ then $x'Fy'$ holds.

This is all that we will need to prove the following lemma.

Lemma 4.4.5 $\quad (f^R)^+(\alpha(x)) = \bigcup_{x_i \in \vec{f}^{+}(x)} \{\alpha(x_i)\}$

Proof: Straightforward. From our modified interpretation for order relations, we know that for every $e_y \in (f^R)^+(\alpha(x))$: $e_y = \alpha(y)$ such that

- *either* $x \ \dot{f} \ x_1, x_1 \ \dot{x}_2, \ldots, x_n \ \dot{f} \ y \in C_s$
- *or* $xfx_1, \ldots, x_{n-1}fx_n, x_nf^+y \in C_s$ where $n \geq 0$

By the definition of $\vec{f}^{+}$ we know that $y \in \vec{f}^{+}(x)$.

Hence $\alpha(y) \in \bigcup_{x_i \in \vec{f}^{+}(x)} \{\alpha(x_i)\}$.

On the other hand, from the definition of $\vec{f}^{+}$, for any $y \in \vec{f}^{+}(x)$:

- *either* $x \ \dot{f} \ x_1, x_1 \ \dot{x}_2, \ldots, x_n \ \dot{f} \ y \in C_s$
- *or* $xfx_1, \ldots, x_{n-1}fx_n, x_nf^+y \in C_s$ where $n \geq 0$

But this implies that $\alpha(y) \in (f^R)^+(\alpha(x))$.

Hence we have the theorem.

Now we show that $\mathcal{R}, \alpha$ models every constraint in C_s. It is enough to consider the following cases:

Case 1 : if $xry \in C_s$ such that $r \in \mathcal{F}i$ then, as illustrated earlier, our modification to the interpretation of r cannot affect the interpretation of xry.

Hence, $(\alpha(x), \alpha(y)) \in r^R$.

Case 2 : From our modified interpretation of relation symbols, it is clear that if $x\ f^+\ y \in C_s$ then $(\alpha(x), \alpha(y)) \in (f^R)^+$.

Hence $\mathcal{R}, \alpha$ satisfies xr^+y.

Case 3: if $p^*(x) \times q^*(y) \dot{\sqsubseteq} F \in C_s$ where F ranges over r, r^+ then taking into account condition 6 of the normal form definition we know that for every $x_i \in \dot{p}^*\ (x)$ and every $y_i \in \dot{q}^*\ (y) : p^*(x_i) \times q^*(y_i) \dot{\sqsubseteq} F' \in C_s$ such that the following dependency holds between F and F':

- if $F \equiv r$ then $F' \equiv r$
- if $F \equiv r^+$ then $F' \in \{r, r^+\}$

This means that:

- $x\ \dot{F}\ y \in C_s$ from condition 6 of the normal form definition. Hence $(\alpha(x), \alpha(y)) \in F^R$.
- it remains to verify that for each $e_x \in (p^R)^+(\alpha(x))$ and $e_y \in (q^R)^+(\alpha(y)) : (e_x, e_y) \in F^R$.

From lemma 4.4.5 we already know that

- $(p^R)^+(\alpha(x)) = \bigcup_{x' \in \vec{p}^+(x)} \{\alpha(x')\}$
- $(q^R)^+(\alpha(y)) = \bigcup_{y' \in \vec{q}^+(y)} \{\alpha(y')\}$

Hence, it is enough to show that:

$p^*(x') \times q^*(y') \dot{\sqsubseteq} F' \in C_s$ for each $x' \in \vec{p}^+\ (x)$ and $y' \in \vec{q}^+\ (y)$ (where the dependency between F and F' is as noted above)

This can easily be verified by induction over the length of the chains $x\ \dot{p}^+\ x_1, \ldots, x_{n-1}\ \dot{p}^+\ x_n \in C_s$ and $y\ \dot{p}^+\ y_1, \ldots, y_{n-1}\ \dot{p}^+\ y_n \in C_s$.

This means that $\mathcal{R}, \alpha$ satisfies every constraint of the form $p^*(x) \times q^*(y) \dot{\sqsubseteq} F$ where F ranges over r, r^+.

Hence in conjunction with the normal form theorem for the language $\mathcal{S}_1$, $\mathcal{R}, \alpha$ satisfies every $\mathcal{T}_1$ constraint. Hence, $\mathcal{R}, \alpha$ satisfies C_s.

Hence, we have the theorem.

Normalisation Rules

We shall employ the following normalisation rules which are in addition to the normalisation rules for the language $\mathcal{S}_1$:

1. $\{x\ r^+\ y\} \cup C_s \longrightarrow C_s$
   if $x\ \dot{r}\ y \in C_s$
2. $\{p^*(x) \times q^*(y) \dot{\sqsubseteq} r,\ p^*(x) \times q^*(y) \dot{\sqsubseteq} r^+\} \cup C_s \longrightarrow \{p^*(x) \times q^*(y) \dot{\sqsubseteq} r\} \cup C_s$
3. $C_s \longrightarrow \{x\ r^+\ z\} \cup C_s$
   if $x\ r^+\ z \notin C_s$, $x\ \dot{r}^+\ y \in C_s$, $y\ \dot{r}^+\ z \in C_s$, $x \not\equiv y$ and $y \not\equiv z$
4. $\{p^*(x) \times q^*(y) \dot{\sqsubseteq} F\} \cup C_s \longrightarrow \{p^*(x) \times q^*(y) \dot{\sqsubseteq} F, xFy\} \cup C_s$
   if F ranges over r, r^+ and $x\ \dot{r}^+\ y \notin C_s$
5. $\{p^*(x) \times q^*(y) \dot{\sqsubseteq} F\} \cup C_s \longrightarrow \{p^*(x) \times q^*(y) \dot{\sqsubseteq} F,\ p^*(x') \times q^*(y) \dot{\sqsubseteq} F\} \cup C_s$
   if $x\ \dot{p}^+\ x' \in C_s$ and $p^*(x') \times q^*(y) \dot{\sqsubseteq} F' \notin C_s$ such that the following dependency holds between F and F':
   - if $F \equiv r$ then $F' \equiv r$
   - if $F \equiv r^+$ then $F' \in \{r, r^+\}$
6. $\{p^*(x) \times q^*(y) \dot{\sqsubseteq} F\} \cup C_s \longrightarrow \{p^*(x) \times q^*(y) \dot{\sqsubseteq} F,\ p^*(x) \times q^*(y') \dot{\sqsubseteq} F\} \cup C_s$
   if $y\ \dot{q}^+\ y' \in C_s$ and $p^*(x) \times q^*(y') \dot{\sqsubseteq} F' \notin C_s$ such that the following dependency holds between F and F':
   - if $F \equiv r$ then $F' \equiv r$
   - if $F \equiv r^+$ then $F' \in \{r, r^+\}$

4.5 Invariance, Termination and Completeness

Theorem 4.5.1 [Invariance] *C_s is a consistent constraint system iff at least one instance of every applicable rule transforms C_s into a consistent constraint system.*

Proof: For the first part, it is clear that each of the above normalisation rules preserves the interpretation of every relation symbol and every variable.

Furthermore, none of the normalisation rules introduce new variables. Hence, every new constraint system computed by the above normalisation rules is equivalent to the original constraint system. Together with the weaker invariance claim for the normalisation rules for $\mathcal{S}_1$ constraint systems we know that the normalisation provedure for $\mathcal{T}_1$ constraint systems is consistency preserving.

This proves the first part.

For the second part, we need to show that every normalisation rule preserves inconsistency. Rules 3 through 6 only add new constraints and do not change existing constraints. Thus these rules does not change the inconsistency of the original constraint system. Hence to verify this claim, it is enough to show that rules 1 and 2 preserve inconsistency. This means that we need to show that the original constraint system is consistent "only if" the transformed constraint system is consistent with respect to these two rules. It is quite clear that this is indeed the case for both of these rules.

This proves the theorem.

We shall not provide a rigorous proof to show that our normalisation rules terminate. It is clear that this is indeed the case since:

1. our rules do not generate new variables and
2. none of the rules apply more than once to the same pair of constraints.

Proposition 4.5.2 [Termination] *There is no infinite chain of rule applications issuing from the normalisation procedure for the language $\mathcal{T}_1$.*

Theorem 4.5.3 [Completeness of Rewriting] *If C_s is a $\mathcal{T}_1$ constraint system then every completion of the normalisation procedure transforms C_s into normal form.*

Proof: To verify the completeness claim we need to show that at least one of the normalisation rules is applicable as long as the constraint system is *not* in normal form. For each of the conditions for normal form given in definition 4.4.2 there is a corresponding normalisation rule.

Hence we have the theorem.

As a corollary to the above results of this section, we have the following.

Corollary 4.5.4 *1. For any $\mathcal{T}_1$ constraint system C_s it is decidable whether C_s is consistent.*

2. Any consistent $\mathcal{T}_1$ constraint system C_s has a solution in the relational graph algebra $\mathcal{R}$.

4.6 Adding Order Constraints to Term languages

So far we have developed a constraint language which provides complex order constraints that can be considered as a possible interpretation of *linear precedence* constraints in a constraint logic setting. In this section we develop a term description language with terms for describing order constraints. This will complete our investigation of term description languages that can be considered as suitable candidates for encoding constraint-based grammars.

Let $\mathcal{ALO}$ be the term language given by the following BNF syntax:

$$S,T \longrightarrow x \mid a \mid c \mid C \mid \exists f:T \mid \forall f:T \mid S \sqcap T \mid S \sqcup T \mid f:\{T_1,\ldots,T_n\} \mid p^*(x) \times q^*(y) \dot{\sqsubseteq} r \mid p^*(x) \times q^*(y) \dot{\sqsubseteq} r^+$$

where $r, f \in \mathcal{F}$, $r \in \mathcal{F}i$ and $f \notin \mathcal{F}i$.

We shall call the term $p^*(x) \times q^*(y) \dot{\sqsubseteq} F$ where F ranges over r, r^+ an *order constraint.*

The language $\mathcal{ALO}$ avoids negations mainly because consistency testing with negative order constraints turns out to be a difficult exercise and we leave this as a future exercise.

Our syntax also restricts transitive relations to order constraints only since the integration of transitive relations in concept languages in a general setting is not the focus of

this Chapter. Our main interest is the integration of linear precedence constraints as a specialisation of order constraints. The integration of transitive relations in concept languages is studied elsewhere in [Baader 91].

The interpretation of *order constraints* is provided by the following definition:

- $[\![p^*(x) \times q^*(y) \dot{\sqsubseteq} F]\!]^{\mathcal{I},\alpha} = \mathcal{U}^I$
  if $(p^I)^*(\alpha(x)) \times (q^I)^*(\alpha(y)) \subseteq F^I$
- $[\![p^*(x) \times q^*(y) \dot{\sqsubseteq} F]\!]^{\mathcal{I},\alpha} = \emptyset$
  if $(p^I)^*(\alpha(x)) \times (q^I)^*(\alpha(y)) \not\subseteq F^I$

The following lemmas can be easily verified.

Lemma 4.6.1 *1. The constraint system* $\{x_0 \dot{\sqsubseteq} (p^*(x) \times q^*(y) \dot{\sqsubseteq} F)\}$ *is consistent iff the constraint system* $\{p^*(x) \times q^*(y) \dot{\sqsubseteq} F\}$ *is consistent.*

2. The constraint system $\{x_0 \dot{\sqsubseteq} X, X \dot{\sqsubseteq} (p^*(x) \times q^*(y) \dot{\sqsubseteq} F)\}$ *is consistent iff the constraint system* $\{p^*(x) \times q^*(y) \dot{\sqsubseteq} F\}$ *is consistent.*

These lemmas provide us with a mechanism to test consistency of $\mathcal{ALO}$ terms and lead to the following additional Stage 3 constraint propagation rules. These rules are in addition to the consistency testing rules for the term language $\mathcal{ALS}$.

Stage 3 : rules

Group 1 : Constraint Simplification rules

All the $\mathcal{T}_1$ *normalisation rules*

Group 3 : Constraint Propagation rules

1. $\{x_0 \dot{\sqsubseteq} (p^*(x) \times q^*(y) \dot{\sqsubseteq} F)\} \cup C_s \longrightarrow \{p^*(x) \times q^*(y) \dot{\sqsubseteq} F\} \cup C_s$
2. $\{x_0 \dot{\sqsubseteq} X, X \dot{\sqsubseteq} (p^*(x) \times q^*(y) \dot{\sqsubseteq} F)\} \cup C_s \longrightarrow \{p^*(x) \times q^*(y) \dot{\sqsubseteq} F\} \cup C_s$

It is quite easy to see that the above rules are *invariant, complete* and *terminating* since these properties are inherited from the constraint language $\mathcal{T}_1$.

From this we conclude the following.

Proposition 4.6.2 [Invariance, Termination, Completeness] *The consistency testing procedure for $\mathcal{ALO}$ constraint systems is invariant, terminating and complete.*

Hence, we have a decision procedure for consistency testing of $\mathcal{ALO}$ terms.

4.7 Summary

In this Chapter, we have explored a novel method for a constraint based solution to the problem of linear precedence constraints.

Our approach can be considered a fairly conservative one in the sense that we have provided a fairly minimal machinery to deal with linear precedence constraints. This is so because terms for expressing linear precedence always have to specify the two variables upon which the constraints need to be satisfied. Furthermore, our semantics for linear precedence terms is clearly a weak one involving either $\mathcal{U}^I$ or $\emptyset$.

Chapter 5

CL-ONE : A Design Study for a Linguistic Formalism

5.1 Introduction

In this Chapter we put to test the theory we have developed in the previous Chapters in the design of a practical knowledge representation language for the representation of linguistic knowledge. This will complete the picture by providing the application-oriented end of the constraint logic we developed in the previous Chapters. Indeed we view the usefulness of constraint logics for NL applications as a necessary prerequisite as opposed to an afterthought and this has always been the overriding theme in this thesis.

We shall call our knowledge representation language - CL-ONE which is an acronym for *Constraint Language ONE*. The philosophy we shall adopt in the design of CL-ONE can be summarised in 3 catch-phrases:

1. Extend (from previous work)
2. Simplicity (of the language)
3. Simple Implementability

Our first design objective states that we build on from existing formalisms. In particular we shall extend previous frameworks such as LOGIN [Aït-Kaci & Nasr 86], STUF

[Dörre & Seiffert 91] and TFS [Emele & Zajac 90, Zajac 90b]. The extensions that we have in mind are principally to do with constraints on set-values and its extension to LP precedence constraints which have so far not been incorporated in linguistically minded knowledge representation frameworks. The logic for set-values and its extension to LP constraints have been dealt with in Chapters 3 and 4 respectively.

Our second design objective is to strive for simplicity of the language. We do not want the language to be complicated for the user to learn or to use nor do we want to incorporate each and every extension of the language $\mathcal{ALS}$ nor the full logic of $\mathcal{ALS}$ in CL-ONE since this will not only be too inefficient but at this stage will also lack much practical value. For this reason we shall focus our target area of application to be the more restricted area of representing *syntactic information* although the logical machinery that we developed in Chapters 2 through 4 will have a much broader application.

Our final design objective is simplicity of the implementation itself. By this we would want to further restrict constructs that are either difficult to implement while serving limited practical value in our intended domain of application or cause considerable loss of efficiency.

5.1.1 Design Parameters

With the above stated design philosophy in mind the next task is to consider what specific features CL-ONE is to possess. The features that are considered important for linguistic knowledge representation tasks are as follows:

1. Sorts
2. Feature Constraints
3. Disjunctions
4. Negations
5. Constraints on Set-values

6. LP constraints
7. Limited Finite Domain Constraints
8. Type System and Multiple Inheritance
9. Relation Typing

The above list takes into account both the current state-of-the-art in linguistic knowledge representation frameworks represented by formalisms such as STUF, TFS and CUF and at the same time makes heavy use of the theoretical framework developed earlier in Chapters 3 through Chapters 4 of this thesis.

In this Chapter we shall use *sorts* to mean sort symbols that do not have a definition and *types* to mean sort symbols that have a definition.

5.2 The Syntax and Semantics of CL-ONE Terms

In this section we provide a picture of the language CL-ONE by defining its syntax and semantics. For most part, CL-ONE is a subset of the language $\mathcal{ALO}$ together with *linear precedence* constraints that we developed in Chapter 4. However, CL-ONE comes with 3 additional features which were not provided in $\mathcal{ALO}$. These are *distributed sort expressions*, *finite domain constraints* and *relation typing definitions*. Each of these additions come with consistency preserving and terminating constraint solving rules and hence we claim that the underlying *terminological component* of CL-ONE is equipped with a *sound*, *complete* and terminating constraint solving machinery. Of course, in addition CL-ONE also has a definitional component which too is *sound* and *complete* borrowing the results from [Höhfeld & Smolka 88]. However, cyclic definitions can lead to non-terminating computations and hence termination in CL-ONE in general is not guaranteed.

The syntax of CL-ONE terms is given by the following BNF definition where $S, T, S_1, \ldots, S_n$ represent CL-ONE terms:

$$\begin{aligned} S,T \longrightarrow\ & Sexp \mid DSexp \mid X \mid S \,\&\, T \mid {\sim}S \mid S : {-}T_1, \ldots, T_n \mid f : T \mid \\ & \exists : S \mid \forall : S \mid \{S_1, \ldots, S_n\} \mid S \prec\prec T \mid n..m \mid I + J \mid S \leq T \end{aligned}$$

where n, m are integers; f is a feature symbol; X ranges over CL-ONE variables; $Sexp$ is a CL-ONE sort expression and $DSexp$ is a CL-ONE distributed sort expression.

The reader is referred to section 5.4 for the syntax of sort expressions. Similarly, section 5.9.1 provides the syntax for distributed sort expressions.

An interpretation structure for CL-ONE terms is provided by a relational algebra $\mathcal{I} = < \mathcal{U}^I, .^I >$ together with a $\mathcal{I}$-assignment α and a context assignment κ such that:

1. $\mathcal{U}^I$ is an arbitrary non-empty set
2. $.^I$ is an interpretation function that maps:
   - every feature symbol $f \in \mathcal{F}$ to a unary partial function $f^I : \mathcal{U}^I \rightharpoonup \mathcal{U}^I$
   - η^I is a binary relation in $\mathcal{U}^I$ *i.e.* $\eta^I \subseteq \mathcal{U}^I \times \mathcal{U}^I$
3. α is a variable assignment function that maps CL-ONE variables to elements in $\mathcal{U}^I$ *i.e.* $\alpha : \mathcal{V} \longrightarrow \mathcal{U}^I$
4. κ is a assignment function that maps disjunction names to integers *i.e.* $\kappa : \mathcal{D} \longrightarrow Int$

The idea here is to provide just a single relational attribute namely η standing for the set-membership relation while every other relation symbol will be interpreted functionally *i.e.* as feature symbols.

The denotation of CL-ONE terms is then provided by a denotation function in the following definitions.

Disjunction names come into play only when we consider the semantics of distributed disjunctions in section 5.9.1 so we shall ignore both disjunction names and the context assignment function κ for the time being.

In the following we assume that $<$ is a feature symbol *i.e.* $< \in \mathcal{F}$ and Int the domain of integers.

1. $[\![X]\!]^I_{\alpha,\kappa} = \{\alpha(X)\}$

2. $[\![S \;\&\; T]\!]^I_{\alpha,\kappa} = [\![S]\!]^I_{\alpha,\kappa} \cap [\![T]\!]^I_{\alpha,\kappa}$

3. $[\![\neg c]\!]^I_{\alpha,\kappa} = \mathcal{U}^I - [\![c]\!]^I_{\alpha,\kappa}$
   where c is a CL-ONE sort symbol or a variable

4. $[\![S : -T_1, \ldots, T_n]\!]^I_{\alpha,\kappa} = [\![S]\!]^I_{\alpha,\kappa}$ if for each $T_i : 1 \leq i \leq n$:$[\![T_i]\!]^I_{\alpha,\kappa} \neq \emptyset$

5. $[\![S : -T_1, \ldots, T_n]\!]^I_{\alpha,\kappa} = \emptyset$ if exists $T_i : 1 \leq i \leq n$:$[\![T_i]\!]^I_{\alpha,\kappa} = \emptyset$

6. $[\![f : T]\!]^I_{\alpha,\kappa} = \{e \in \mathcal{U}^I \mid f^I(e) \in [\![T]\!]^I_{\alpha,\kappa}\}$

7. $[\![\exists : S]\!]^I_{\alpha,\kappa} = \{e \in \mathcal{U}^I \mid \exists e' \in [\![S]\!]^I_{\alpha,\kappa} : (e, e') \in \eta^I\}$

8. $[\![\forall : S]\!]^I_{\alpha,\kappa} = \{e \in \mathcal{U}^I \mid \forall e' : (e, e') \in \eta \Rightarrow e' \in [\![S]\!]^I_{\alpha,\kappa}\}$

9. $[\![\{S_1, \ldots, S_n\}]\!]^I_{\alpha,\kappa} = \{e \in \mathcal{U}^I \mid \eta^I(e) = \{e_1, \ldots, e_n\} \wedge e_1 \in [\![S_1]\!]^I_{\alpha,\kappa} \wedge \ldots \wedge e_1 \in [\![S_n]\!]^I_{\alpha,\kappa}\}$

10. $[\![S \prec\prec T]\!]^I_{\alpha,\kappa} = \mathcal{U}^I$ if $(\eta^I)^*(\alpha(x)) \times (\eta^I)^*(\alpha(y)) \subseteq (<^I)^+$

11. $[\![S \prec\prec T]\!]^I_{\alpha,\kappa} = \emptyset$ if $(\eta^I)^*(\alpha(x)) \times (\eta^I)^*(\alpha(y)) \not\subseteq (<^I)^+$

12. $[\![n..m]\!]^I_{\alpha,\kappa} = \{e \in \mathcal{U}^I \mid e \in Int \wedge n \leq e \leq m\}$

13. $[\![I + J]\!]^I_{\alpha,\kappa} = \{e \in \mathcal{U}^I \mid e \in Int \wedge \exists e_s \in [\![I]\!]^I_{\alpha,\kappa} \wedge \exists e_t \in [\![J]\!]^I_{\alpha,\kappa} : e = e_s + e_t\}$

14. $[\![S \leq T]\!]^I_{\alpha,\kappa} = \mathcal{U}^I$
    if $[\![S]\!]^I_{\alpha,\kappa} \subseteq Int$, $[\![T]\!]^I_{\alpha,\kappa} \subseteq Int$, $\exists e \in [\![S]\!]^I_{\alpha,\kappa}, \exists e' \in [\![T]\!]^I_{\alpha,\kappa} : e \leq e'$.

15. $[\![S \leq T]\!]^I_{\alpha,\kappa} = \emptyset$
    otherwise

The semantics of of sort expressions and distributed sort expressions are covered separately in sections 5.4 and 5.9.1 respectively.

In the following sections, we explore each of the CL-ONE design features considered in section 5.1.1 to provide an expository account of each of these features and to illustrate the factors that influenced our design decisions.

5.3 Sort System

Sorts provide a convenient generalisation over *atoms, constants* and *primitive concepts* of the language $\mathcal{ALV}$ by adding the notion of an *isa/ako* hierarchy. For instance the term $(gender : sg)$ states that the value of the *gender* label is represented by the sort sg. On the other hand $(gender : sg \sqcup pl)$ states that the value of the *gender* label could be either sg or pl. In both these cases sg and pl are assumed to be *atomic.*

However, the expressiveness of sorts arises as it is also possible to partially order sorts in a hierarchy - for instance as shown in 5.14.

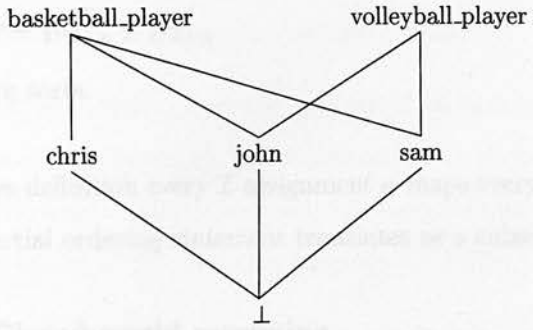


Figure 5.14: A partially ordered Sort hierarchy

We can represent the information in figure 5.14 by the statements such as:

(61) $chris \leq basketball\_player.$

$john \leq basketball\_player.$

$sam \leq basketball\_player.$

$john \leq volleyball\_player.$

$sam \leq volleyball\_player.$

The *partial ordering* statements capture the notion of an *isa* hierarchy. Thus, for instance, the statement $chris \leq basketball\_player$ state that *chris isa basketball_player*. *Sorts* are modelled as *set-denoting* expressions that denote *subsets* of the *universe* $\mathcal{U}^I$.

A CL-ONE *sort system* is a collection of sort symbols together with a partial order on the collection. We assume that the partial order has a unique $\top$ symbol and a $\bot$

symbol.

We say that a relational algebra and an $\mathcal{I}$-assignment α satisfies a sort system if the following conditions are satisfied for each sort symbol a and each partial ordering statement $a \leq b$:

(62)
$$\alpha(a) \subseteq \mathcal{U}^I$$
$$[\![\top]\!]^I_{\alpha,\kappa} = \mathcal{U}^I$$
$$[\![\bot]\!]^I_{\alpha,\kappa} = \emptyset$$
$$[\![a]\!]^I_{\alpha,\kappa} = \alpha(a)$$
$$a \leq b \Longleftrightarrow [\![a]\!]^I_{\alpha,\kappa} \subseteq [\![b]\!]^I_{\alpha,\kappa}$$

where a, b are sorts.

According to the above definition every $\mathcal{I}$-assignment α maps every sort to a subset of $\mathcal{U}^I$ such that every partial ordering statement translates as a subset relation.

5.3.1 Open *vs.* Closed-world reasoning

The sort system of a knowledge representation language is subject to variation along the following parameter:

1. closed world semantics
2. open world semantics

In the above example in figure 5.14, suppose we want to find out "somebody who is both a basketball player and a volleyball player", a possible query could be

basketball_player & *volley_player*

and one possible reply would be

john ⊔ *sam*

Clearly, to arrive at the above answer the semantic condition given in (62) is not enough, instead an implicit *closed world reasoning* is being used. This reasoning has to be encoded in the semantics of *sorts*.

One way to arrive at the above answer is to interpret the denotation of $s \ \& \ t$ where s and t are sort symbols as being equivalent to the denotation of the greatest lower bound of s and t (*i.e.* $glb(a,b)$) in the sort hierarchy *i.e.*

(63) $\quad [\![s \ \& \ t]\!]^I_{\alpha,\kappa} = [\![glb(s,t)]\!]^I_{\alpha,\kappa}$ **glb semantics**

The *glb semantics* is essentially a form of *closed world reasoning* since it assumes that there is total information on the intersection of sort symbols.

Note also that the partial ordering depicted in figure 5.14 is not a lattice, hence the definition of the *greatest lower bound*(glb) operation on lattice elements must be generalised to arbitrary partial order. This is essentially the same measure taken in the design of the sort system for the logic programming language LOGIN [Aït-Kaci & Nasr 86].

Instead of adopting a *closed world semantics* we can equally adopt an *open world semantics*. In an *open world semantics* the semantic property (62) of partial orders hold, but there is no *glb semantics* involved and hence with this semantics it does *not* follow that:

$$[\![basketball\_player \ \& \ volleyball\_player]\!]^I_{\alpha,\kappa} = [\![john \sqcup sam]\!]^I_{\alpha,\kappa}$$

This is so for two reasons:

1. It allows (for the possibility) for the interpretation of *basketball_player* to contain players other than *chris*, *john* and *sam*, say for instance say *Ian* and *Alan*. And similarly for the interpretation of *volleyball_player*. Note, that allowing this possibility does not violate the semantic condition given in (62).

2. It allows for the possibility that the intersection of any two sorts $[\![a]\!]^I_{\alpha,\kappa} \cap [\![b]\!]^I_{\alpha,\kappa}$ be smaller than $[\![glb(a,b)]\!]^I_{\alpha,\kappa}$ *i.e.* $[\![a]\!]^I_{\alpha,\kappa} \cap [\![b]\!]^I_{\alpha,\kappa} \subset [\![glb(a,b)]\!]^I_{\alpha,\kappa}$ would be permitted by an open world semantics. Note again that allowing this possibility does not violate the semantic condition given in (62).

The interesting aspect of the glb semantics is that although it eliminates the second of the above two possibilities, it does not prevent the first possibility. Thus, allowing for the possibility of other players for instance say *Ian* and *Alan* to be either basketball or volleyball players is still fine even with the glb semantics.

This raises the question as to whether it is possible to come up with a stronger closed world reasoning mechanism that eliminates this extra degree of freedom permitted by the glb semantics. Indeed this is possible if we add an extra condition to the way in which sorts are interpreted.

For any given sort system and a sort symbol s let $LB(s)$ denote the set consisting of the lower bounds of s given by:

$$LB(s) = \{t \mid t \leq s\}$$

Let $SLB(s)$ denote the set:

$$SLB(s) = LB(s) - \{s\}$$

We can then provide a stronger interpretation of a sort system by the following additional condition.

(64) if $SLB(s) \neq \{\bot\}$ then:

$$[\![s]\!]^I_{\alpha,\kappa} = \bigcup_{t \in SLB(s)} [\![t]\!]^I_{\alpha,\kappa}$$ **strong closed-world semantics**

The restriction "if $SLB(s) \neq \{\bot\}$ then ..." turns out to be necessary since otherwise in a sort hierarchy such as the one depicted in *fig.* 5.15 both $[\![s]\!]^I_{\alpha,\kappa} = \emptyset$ and $[\![t]\!]^I_{\alpha,\kappa} = \emptyset$.

In effect, the *strong closed world semantics* states that the interpretation of a sort is completely determined by the interpretation of its subsorts. Thus in the example depicted in figure 5.15 the interpretation of $\top$ is given by:

$$[\![\top]\!]^I_{\alpha,\kappa} = [\![s]\!]^I_{\alpha,\kappa} \cup [\![t]\!]^I_{\alpha,\kappa}$$

whereas with the *glb semantics* the above condition is not necessary.

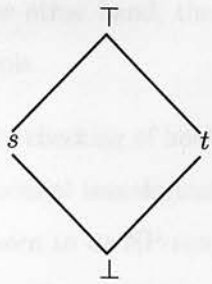


Figure 5.15:

The above discussion on closed *vs.* open-world interpretation of sorts shows that the interpretation of sort systems is subject to the following parameter for variation:

1. *open world semantics*
2. *glb semantics*
3. *strong closed world semantics*

Following the original work by *Hassan Aït-Kaci* [Aït-Kaci 84] the glb semantics is now fairly standard and turns out to be quite useful in practice. However, the need for a strong closed world semantics in natural language applications is less well motivated. This is not to say that there are no formalisms that employ a strong closed world semantics. For instance, the TFS formalism [Zajac 92] employs a strong closed world semantics. However, the motivation for employing a strong closed world semantics in TFS appears to be due to its intended database applications as opposed to natural language applications. Whichever the case may be, we are not convinced for the need of a strong closed world reasoning apparatus for NL applications. For this reason, the design decision adopted in CL-ONE is to opt for a *glb semantics*.

5.4 Boolean constraints

While whether to opt for a closed or open world semantics for computing the conjunction of sort symbols represents one parameter for variation for order-sorted sort systems, another source of variation arises when we consider what form of sort expressions are permitted. For instance, the sort system provided in STUF provides

a full propositional syntax. On the other hand, the language LOGIN provides only disjunctions over atomic sort symbols.

One can easily show that consistency checking of boolean sort expressions is a *NP-hard* problem by demonstrating a polynomial transformation of the satisfiability problem for propositional logic which is known to be NP-complete [Garey & Johnson 79] into a satisfiability problem for boolean sort expressions.

Let ϕ be any given propositional formula for which satisfiability is to be determined. Let $atoms(\phi)$ denote the set of atomic literals occurring in ϕ. To construct this transformation it is enough to construct a sort lattice such that for every subset of atoms $\psi \subseteq atoms(\phi)$ there exists a unique sort a_ψ which is the greatest lower bound of ψ. This will ensure that conjunctions of positive atoms does not result in an inconsistency since the existence of the glb of conjunctions of positive atoms is guaranteed by construction. Furthermore this construction will allow negation to be treated appropriately. This demonstrates that consistency checking of boolean sort expressions is *NP-hard.*

On the other hand, the converse transformation can be established by translating every sort symbol s (in a given boolean sort expression) by the disjunction of its lowers bound *i.e.* $\sqcup LB(s)$. This has the effect that the propositional formula $\sqcup LB(s)$ & $\neg(\sqcup LB(t))$ is inconsistent just in case $s \leq t$. This demonstrates that consistency testing of boolean sort expressions is *NP-easy.* Combining the two results, we know that consistency testing of boolean sort expressions is *NP-complete.*

This shows that consistency testing of boolean sort expression is exponential in the worst case even if efficient bit-vector implementation techniques are employed such as in the STUF formalism following the work of [Shensa 89].

The design decision adopted in CL-ONE is to employ a slightly restricted propositional syntax that permits *disjunctions, conjunctions* and importantly *negations* of sort symbols in such a way as to permit a compact normal form and efficient *quadratic-time* worst case computational complexity. We believe that a *quasi-linear* time complexity can be achieved by adopting efficient bit-vector encoding techniques.

We shall call propositional CL-ONE sort expressions *boolean constraints.* A CL-ONE

boolean constraint is a propositional formula where *sort* symbols are primitive and a formula can be formed by using the usual propositional connectives and has to obey the following restricted syntax:

$$Sexp \longrightarrow s \mid \neg s \mid P \;\&\; N$$

$$P \longrightarrow s \mid P_1 \;\&\; P_2 \mid P_1 \sqcup P_2$$

$$N \longrightarrow \neg s \mid N_1 \;\&\; N_2$$

The syntax permits arbitrary conjunctions and disjunctions of sort symbols but negated sort symbols are not allowed to appear within the scope of a disjunction. Also, negation is only allowed in an atomic expression. Hence, although $s \;\&\; t \;\&\; \neg u$ is permitted $\neg s \sqcup \neg t$ is not permitted.

In the NL applications that we have encountered so far, there has not been a single instance where a full propositional syntax was essential. This justifies our choice of the above restricted syntax since not only does it permit a fairly simple implementation, as we shall show, it has a provable quadratic worst case complexity which can be further improved by adopting efficient bit-vector implementation techniques.

The interpretation of *sort expressions* is given by the following definitions:

1. $[\![s \;\&\; t]\!]^I_{\alpha,\kappa} = [\![s]\!]^I_{\alpha,\kappa} \cap [\![t]\!]^I_{\alpha,\kappa}$
2. $[\![s \sqcup t]\!]^I_{\alpha,\kappa} = [\![s]\!]^I_{\alpha,\kappa} \cup [\![t]\!]^I_{\alpha,\kappa}$
3. $[\![\neg s]\!]^I_{\alpha,\kappa} = \mathcal{U}^{\mathcal{I}} - [\![s]\!]^I_{\alpha,\kappa}$

Note that furthermore $[\![s \;\&\; t]\!]^I_{\alpha,\kappa} = [\![glb(s,t)]\!]^I_{\alpha,\kappa}$ is already fixed by our choice of the glb semantics.

Normalisation of Boolean Constraints

Next, we shall present a compact representation for CL-ONE sort constraints and show that the normalisation of boolean constraints can be performed in quadratic time in the worst case.

In the following let $\sqcap\{s_1, \ldots, s_n\}$ represent a (abstract) *datastructure* for representing the *conjunction* of the atomic sort symbols s_1 through s_n. Similarly, let $\sqcup\{s_1, \ldots, s_n\}$ represent a *datastructure* for representing the *disjunction* of the atomic sort symbols. And let $\neg\{s_1, \ldots, s_n\}$ represent a *datastructure* for representing the *conjunction* of the *negations* of the atomic sort symbols.

In other words, we have the following definitions:

(65) Let $\sqcap\{s_1, \ldots, s_n\} =_{def} s_1 \ \& \ \ldots \ \& \ s_n$.

(66) Let, $\sqcup\{s_1, \ldots, s_n\} =_{def} s_1 \sqcup \ \ldots \sqcup s_n$.

(67) Similarly let, $\neg\{s_1, \ldots, s_n\} =_{def} \neg s_1 \sqcap \ \ldots \sqcap \neg s_n$.

Furthermore, let $\sqcup P/\neg N =_{def} \sqcup P \ \& \ \neg N$ where $\sqcup P$ and $\neg N$ are as in (65) and (67) respectively.

By overloading the notation we shall use P/N to mean $\sqcup P/\neg N$.

We shall show that the syntax of CL-ONE sort expressions permits a compact representation of the form P/N which is as defined above.

We say that P/N is in normal form if it satisfies the following additional property:

(68) for each $x \in P$, there is **no** $y \in N$ *s.t.* $x \leq y$.

Theorem 5.4.1 *Every CL-ONE sort expression can be transformed into an equivalent expression of the form P/N in normal form in exponential time.*

Proof: First note that every CL-ONE boolean constraint can be equivalently written as $P \ \& \ N$ where:

1. P is either $\top$ or contains only conjunctions and disjunctions of sort symbols but no negations and
2. N is either $\top$ or is equivalent to $\neg N'$.

Since $\neg N'$ is already a normal form for N it remains to show that P can be transformed into a normal form. We do this in two steps:

Step 1: Translate P into its DNF and replace every conjunct by its *glb*. This eliminates every conjunct in P and hence can equivalently written as $\sqcup P'$.

Step 2: After the end of Step 1, P & N has the form P'/N'. To translate P'/N' into normal form we translate P'/N' into $(P' - N')/N'$ where $P - N$ is an operation defined by:

$$P - N = \{x \in P \mid \textit{for each } y \in N : x \not\leq y\}$$

Note the fact that P'/N' is equivalent to $(P' - N')/N'$ follows from our semantics.

Since expansion to DNF is potentially exponential, we have exponential complexity in the worst case.

Theorem 5.4.2 *The conjunction P_1/N_1 & P_2/N_2 of normal sort descriptions P_1/N_1 and P_2/N_2 can be normalised in time proportional to n^2 where n is the number of sort symbols in the sort lattice.*

Proof: Given normal sort descriptions, P_1/N_1 and P_2/N_2 the conjunction of these descriptions $P/N = P_1/N_1$ & P_2/N_2 can be computed as follows :

$$P/N = (P - R)/R \text{ where } R = N_1 \bigcup N_2 \text{ and } P = glb(P_1, P_2)$$

This operation essentially translates P_1 & P_2 into DNF.

To calculate the time needed to normalise two sort descriptions, lets assume that we are given P_1/N_1 and P_2/N_2. If we assume that the $glb(x, y)$ operation[1] (where x and y are primitive sorts) takes a time proportional to some constant g then the time required to compute $P = glb(P_1, P_2)$ is $O(g \times n \times n)$ where n the size of the sort hierarchy is an upper bound for both $\mid P_1 \mid$ and $\mid P_2 \mid$. The operation $P - R$ will require $O(k \times n \times n)$ since n is an upper bound for the size P and R. Hence the total time required is $O(gn^2 + kn^2) = O(n^2)$. Thus, our sort descriptions can be normalised in quadratic time. Moreover, our representation is well suited to a fairly simple prolog implementation.

[1]For instance to achieve a constant time, it is possible to precompute all possible *glb* operations on primitive sorts in advance and store it in a table. But, a better approach would be to build this table lazily.

We claim that the restricted syntax is good enough for most linguistic applications and hence compares favourably with systems that provide a full propositional syntax such as in STUF [Dörre & Seiffert 91]. Since we know that the decision problem for the full calculus is NP-complete, there are no known polynomial decision algorithms. Furthermore, employing efficient lattice encoding techniques [Aït-Kaci *et al* 85] [Shensa 89] for encoding our representation P/N would result in a linear time normalisation algorithm.

5.5 Feature Constraints

The ability to represent feature structures is perhaps the single most important factor in a natural language application. Feature selection is achieved in CL-ONE by using the term $f : T$.

Path equations and path disequations [Kasper & Rounds 86] are not available in CL-ONE. However, these can be alternatively encoded in CL-ONE *via* variables. For instance, the path equation:

$$agr = head : arg$$

can be alternatively encoded in CL-ONE by:

$$agr : X \ \& \ head : (agr : X)$$

Cyclic co-references present no problems to CL-ONE and the user is free to represent cyclic structures.

5.6 Set Descriptions

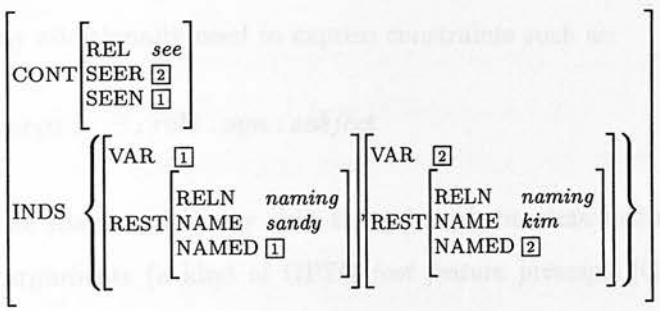
Set descriptions provide a mechanism for modelling complex linguistic structures such as *semantic indices* in HPSG [Pollard & Sag 87, Pollard & Sag 92], *word-ordering constraints* (see Chapter 4), *situation theoretic structures* [Rounds 88], the modelling of *thematic grids* (see Chapter 6) and the modelling of *subcategorisation frames* ([Bes & Gardent 89] and see Chapter 6). None of these complex structures can be modelled within feature logic (say *e.g.* [Smolka 88] or [Kasper & Rounds 86]). The central

reason is that features are interpreted as *partial functions* whereas set descriptions need to be interpreted *relationally.*

Consider the following examples which make use of set descriptions.

The first example is an HPSG AVM (*attribute-value matrix*) definition for the semantics of the sentence *Kim saw Sandy* [Pollard & Sag 87, *pp. 104*]:

(69)



This can be encoded in linear CL-ONE syntax as follows:

(70)

cont : (*rel* : *see* &
seer : *X* &
seen : *Y*
) &
inds : { *var* : *X* &
rest : (*reln* : *naming* &
name : *sandy* &
named : *X*
),
var : *Y* &
rest : (*reln* : *naming* &
name : *kim* &
named : *Y*
)
}

The second example is the modelling of the thematic grid for the verb *promise/persuade* given in CL-ONE syntax:

(71)

args : { *role* : *theta* : *agent*,
role : *theta* : *patient*,
role : *theta* : *proposition*
}

However, just being able to represent set descriptions such as the above is not very useful. This is so, since most of the time we not only want to describe sets but we also need to specify constraints on them. For instance, in example (71) we may want to additionally specify that at least one of the thematic arguments has to be realised as a *subject.*

Thus to express the constraint that at least one of the thematic arguments in example (71) we may additionally need to express constraints such as:

(72) $args: \quad \exists : role : syn : subject$

Similarly, we may need to say that the value of the *slash* feature must be identical across all arguments (a kind of GPSG *foot feature principle* [Gazdar *et al* 85]) by a constraint such as:

(73) $args: \quad \forall : slash : X$

Thus, *existential* and *universal* constraints provide an elegant way to express complex constraints on set descriptions.

CL-ONE supports the following constraints on sets (this is excluding the LP constraints which also operate on sets but are described separately in the next section):

- HPSG style partial set descriptions
- Set-membership Constraints
- For-all construct

The syntax of each of these constructs and its corresponding syntax in the language $\mathcal{ALO}$ (see Chapter 3) is provided in the following:

CL-ONE syntax	$\mathcal{ALO}$ **Syntax**	
$\{T_1, \ldots, T_n\}$	$\eta : \{T_1, \ldots, T_n\}$	*(Set Description)*
$\exists : T$	$\exists \eta : T$	*(Set-membership constraint)*
$\forall : T$	$\forall \eta : T$	*(Forall Constraint)*

Thus, CL-ONE provides the full complement of constraints for describing sets which are available in the language $\mathcal{ALO}$. Each of these constraints turn out to be essential for the kind of linguistic applications that we deal with in Chapter 6.

For a detailed technical exposition on set descriptions, the reader is referred to Chapter 3 where the complete logical semantics and the constraint solving machinery are covered.

The provision of set descriptions and the associated constraints of set descriptions including LP constraints (*see* Section 5.8) is an important first in CL-ONE since none of the current generation feature logical based formalisms (*e.g* STUF [Dörre & Seiffert 91], TFS [Zajac 90b], CUF [Eisele & Dörre 90]) support set descriptions.

Secondly, the provision of set descriptions in CL-ONE achieves yet another important goal in knowledge representation : that of bringing closer the area of feature logic based systems and KL-ONE type terminological languages since both can be considered as *attributive description languages* in the sense of [Nebel & Smolka 89] (since in feature logics the *attributes* are known as *features* while in terminological languages they are known as *roles*). In fact CL-ONE can be considered to be a terminological language designed with natural language applications in mind. CL-ONE represents the beginnings of an *integrated* knowledge representation language that fully supports unification-based grammars.

5.7 Finite domain constraints

Finite domains are a generalisation over ordinary integers that allow a range of values to be specified for a given variable as opposed to a single distinct value. An ordinary integer restriction for a given variable X will be a constraint of the form:

$$X \ \& \ Int$$

which states that the variable X ranges over *integers*. With finite domains one would allow constraints of the form:

X & [0..200]

which states that the value of X must be between 0 and 200. Finite domain constraints could be used in addition to the usual typing constraints of the form X & Int.

Finite domains were introduced in [Van Hentenryck 89] [Van Hentenryck & Dincbas 86] [Van Hentenryck & Dincbas 87] and later included in the constraint logic programming language CHIP [Dincbas *et al* 88]. The techniques we employ for including finite domains in CL-ONE are adapted from the ones that are found in the above mentioned literature.

The principal application of finite domains within the linguistic application we survey is for proto-role selection (see Chapter 6). However, more generally, finite domains have other applications in natural language, for instance for the representation of numeric quantifiers, dates *etc.*

In other to motivate the kind of constraint solving rules needed to deal with finite domain constraints we shall look at some examples of constraint satisfaction involving finite domains.

Consider the conjunction of finite domains as illustrated by the following example:

[1..20] & [5..30]

Simplification of the above constraint would result in the following normalised constraint:

[5..20]

In CL-ONE we will permit inclusion of simple arithmetic linear constraints of the form:

[1..20] + [5..30]

Simplification of the above constraint produces:

[6..50]

More generally, CL-ONE would permit *equality* constraints of the form:

$$([1..20] + [5..30]) \ \& \ [10..85]$$

which gets simplified to:

$$[10..60]$$

and *inequality* constraints of the form:

$$([1..20] + [5..30]) \leq [4..60]$$

which gets simplified to:

$$[6..50] \leq [6..60]$$

Similarly the constraint:

$$[5..20] \leq [6..15]$$

would be simplified to:

$$[5..15] \leq [6..15]$$

These complete the repertoire of finite domain constraints available in CL-ONE.

5.7.1 Constraint solving with finite domains

In this section we rigorously state the constraint solving rules needed for consistency checking of finite domain constraints.

(BotCheck) $$\frac{\{x : [I..J]\} \cup C_s}{\{x : \bot\} \cup C_s}$$
if $J > I$

(Add) $$\frac{\{x : ([I_1..J_1] + \ldots + [I_n..J_n])\} \cup C_s}{\{x : [I_1 + \ldots + I_n \,..\, J_1 + \ldots + J_n]\} \cup C_s}$$

(Conj) $$\frac{\{x : [I..J], x : [K..L]\} \cup C_s}{\{x : [M..N]\} \cup C_s}$$
where $M = max(I, K)$ and $N = min(J, L)$

(Less) $$\frac{\{x : [I..J], y : [K..L], x \leq y\} \cup C_s}{\{x : [I..M], y : [N..L], x \leq y\} \cup C_s}$$
where $M = min(J, L)$ and $N = max(I, K)$

Rule BotCheck is intended to check for inconsistency between the lower and upper domain ranges and reports an inconsistency if the upper range is *less* than the lower range.

Rule Add is intended to normalise constraints involving addition of finite domains.

Rule Conj normalises multiple finite domain constraints attached to the same variable.

Rule Less propagates constraints involving inequality constraints.

The above constraint solving rules provide a fairly minimal support for dealing with finite domains. Note that CL-ONE does not provide support for either enumerated constraints of the form:

$$X : [4, 7, 9, 11..40]$$

or *non-linear arithmetic constraints.*

5.8 Linear Precedence Constraints

Linear precedence constraints provide a means to express partial constraints on the order of words. Our approach to LP constraints generalises the approach used in GPSG and HPSG in that LP constraints need not be stated across the board as is done in GPSG, instead they can be stated as part of a lexical specification. This has the advantage that word ordering variation can be made sensitive to *local* syntactic/thematic/semantic information which was not possible with the GPSG LP mechanism.

Both the logical and constraint solving machinery for LP constraints have been covered in Chapter 4. In this section, we only present the CL-ONE syntax for LP constraints.

LP constraints are stated in CL-ONE by a single binary constraint of the form:

$$T_1 \ll T_2$$

where T_1 and T_2 are CL-ONE terms.

As an example, the following LP constraint:

(74) $\{john, mary\} \ll \{likes\}$

enforces the condition that both *john* and *mary* must precede *likes*. Thus the following constraints are compatible with the above constraint:

$john \ll mary \;\&\; mary \ll likes$

$mary \ll john \;\&\; john \ll likes$

However, the following constraints on the other hand are incompatible with the constraint in (74):

$john \ll likes \;\&\; likes \ll mary$

$mary \ll likes \;\&\; likes \ll john$

$likes \ll john \;\&\; john \ll mary$

CL-ONE syntax also permits nested set-descriptions, set-membership constraints and feature terms as part of LP constraints. Thus the following are syntactically well-formed CL-ONE expressions:

(75) $\{john, \{mary\}\} \ll \{likes\}$

(76) $\exists : john \ll \{likes, mary\}$

(77) $\exists : john \ll \{likes, \{mary\}\}$

The constraint propagation rules of CL-ONE which are based on the language ALO(see Chapter 4) will generate the following constraints for each of the examples in (75) through (77).

(78) $john \ll likes$ generated from (75)
$mary \ll likes$

(79) $john \ll likes$ generated from (76)
$john \ll mary$

(80) $john \ll likes$ generated from (77)
$john \ll mary$

Note that our constraint solving rules take into account the transitivity of linear precedence constraints over the set-membership relation η.

These examples illustrate that CL-ONE provides considerable flexibility in the specification of LP constraints.

5.9 Disjunctive Constraints

Often a purely conjunctive specification may turn out to be insufficient for certain knowledge representation tasks. For instance, to state that the grammatical case can take values acc, nom or dat, we may write the disjunctive constraint:

$$case : acc \sqcup nom \sqcup dat$$

However, disjunctive constraints are also a considerable source of inefficiency. We know from [Kasper 87] and [Smolka 88] that the provision of disjunctions in feature logic renders the consistency problem NP-complete. Thus it is desirable to restrict the usage of disjunctions to eliminate sources of inefficiency.

Since our design goal was to achieve simplicity in implementation and achieve a reasonably efficient implementation, our decision was to carefully analyse the linguistic requirements of disjunctive constraints and to provide a simple but sufficiently adequate form of disjunctive constraints thus striving for a maximum possible average case efficiency.

The principal linguistic applications of disjunctive constraints in computational linguistics are summarised in [Dörre *et al* 90]. Some representative examples taken from [Dörre *et al* 90] are provided in (81), (82) and (83).

(81)
$$\begin{array}{l}(number : sg \;\;\&\;\; (\;\; (gender : masc \;\&\; case : nom) \\ \qquad\qquad\qquad\qquad\qquad\qquad \sqcup \\ \qquad\qquad\qquad\qquad\qquad\qquad (gender : fem \;\&\; case : (gen \sqcup dat)) \\ \qquad\qquad\qquad\qquad\qquad\quad) \\ ) \\ \sqcup \\ (number : pl \;\&\; case : gen)\end{array}$$

(82)
$$\begin{array}{l}(gender : fem \;\sqcup\; number : pl) \;\& \\ case : (nom \sqcup acc))\end{array}$$

(83)
$$\begin{array}{l}gender : neut \;\;\&\;\; (\;\; (number : sg \;\&\; case : (nom \sqcup dat \sqcup acc)) \\ \qquad\qquad\qquad\qquad\qquad\qquad \sqcup \\ \qquad\qquad\qquad\qquad\qquad\qquad (number : pl) \\ \qquad\qquad\qquad\qquad\qquad\quad)\end{array}$$

Our analysis of these examples reveals that each of the above cases can be alternatively represented by using a restricted form of *distributed disjunctions* [Dörre & Eisele 89, Dörre & Eisele 90]. Consider the following alternative linear encoding of example (81) (in CL-ONE syntax).

(84)
$$\begin{array}{l}number : D1/[sg, pl] \;\& \\ gender : D1/[D2/[masc, fem], \top] \;\& \\ case : D1/[D2/[nom, D3/[gen, dat]], gen]\end{array}$$

In the above example the variables $D1, D2$ and $D3$ stand for disjunction names and the disjunct such as $D1/[sg, pl]$ represents a labelled choice point. This means that either the first argument of $D1$ has to be chosen *globally* across the whole description or the second argument of $D1$ has to be chosen globally. Thus when the value of the *number* feature is sg then the value of the *gender* feature must be either $masc$ or fem.

The remaining two examples in (82) and (83) can be alternatively encoded as follows:

(85)
$$\begin{array}{l}gender : D1/[fem, \top] \;\& \\ number : D1/[\top, pl] \;\& \\ case : D2/[nom, acc]\end{array}$$

(86)
$$\begin{array}{l}gender : neut \;\& \\ number : D1/[sg, pl] \;\& \\ case : D1/[D2/[nom, dat, acc], \top]\end{array}$$

Thus *distributed disjunctions* offer a concise notation for expressing disjunctive constraints. In [Dörre & Eisele 89, Dörre & Eisele 90] it is shown that the constraint solving machinery can be optimised for handling disjunctions if *distributed disjunctions* are used instead of conventional disjunctions. Of course this optimisation only improves the average case performance but the worst case performance remains *non-polynomial.*

The design decision taken was to provide a restricted form of *distributed disjunctions* which restricts the arguments of distributed disjunctions to only CL-ONE *sort expressions.* We shall call our version of distributed disjunctions - *distributed sort expressions.* This will permit all the above examples to be encoded in CL-ONE.

However, this will also mean that disjunctions of a more general nature such as the example in (87) taken from [Dörre *et al* 90] cannot be handled.

(87)
$$\begin{array}{l} phon : eats \ \& \\ syn : loc : subcat : (< X1 \ \& \ NP1, X2 \ \& \ NP2 > \sqcup < X2 \ \& \ NP2 >) \ \& \\ sem : cont : (\quad reln : see \ \& \\ \qquad\qquad\quad seer : X2 \ \& \\ \qquad\qquad\quad seen : X1) \end{array}$$

In CL-ONE we resort to disjunctions in the *definitional mechanism* (see section 5.11) to cope with examples such as above. A simple Prolog-style backtracking strategy would be employed to cope with such cases.

5.9.1 Distributed Sort Expressions in CL-ONE

Distributed sort expressions in CL-ONE obey the following *restricted* BNF syntax:

$$DSexp \longrightarrow X/[DSexp_1, \ldots, DSexp_n] \mid Sexp$$

where $DSexp$ and $DSexp_1, \ldots, DSexp_n$ are *distributed sort expressions*; X is a *disjunction name* and $Sexp$ is a CL-ONE *sort expression.* It is further required that the set of *disjunction names* be distinct from the set of CL-ONE *variable names.*

The semantics of CL-ONE distributed sort expressions is given by:

$$[\![D/[S_1, \ldots, S_n]]\!]^I_{\alpha,\kappa} = [\![S_i]\!]^I_{\alpha,\kappa} \text{ whenever } \kappa(D) = i.$$

This states that the value of a distributed sort expression is dependent upon the context which is determined by κ.

5.9.2 Normalisation of CL-ONE distributed sort expressions

Normalisation of CL-ONE distributed sort expressions is carried out by the following rules given in *natural deduction* style.

In the following $\phi_1, \ldots, \phi_n$ and $\psi_1, \ldots, \psi_m$ range over *disjunctive sort expressions.*

(88)
$$\frac{X/[\phi_1, \ldots, \phi_n] \;\&\; Y/[\psi_1, \ldots, \psi_m]}{\begin{array}{l} X/[\quad Y/[\phi_1 \& \psi_1, \ldots, \phi_1 \& \psi_m], \\ \qquad Y/[\phi_2 \& \psi_1, \ldots, \phi_2 \& \psi_m], \\ \qquad \ldots, \\ \qquad Y/[\phi_n \& \psi_1, \ldots, \phi_n \& \psi_m] \\ \quad] \end{array}} \quad \text{where } X \neq Y$$

(89)
$$\frac{X/[\phi_1, \ldots, \phi_n] \;,\; X/[\psi_1, \ldots, \psi_n]}{X/[\phi_1 \& \psi_1, \ldots, \phi_n \& \psi_n]}$$

(90)
$$\frac{X/[\phi_1, \ldots, \phi_n] \;,\; X/[\psi_1, \ldots, \psi_m]}{X : \bot} \quad \text{where } n \neq m$$

(91)
$$\frac{X/[X_1 : \bot, \ldots, X_n : \bot]}{X : \bot}$$

(92)
$$\frac{X/[\ldots, Y/[\phi_1, \ldots, \phi_n] \;, \ldots], \; Y : \bot}{X/[\ldots, Y : \bot, \ldots] \;,\; Y : \bot}$$

The rule in (88) recursively propagates disjunctive constraints by successively multiplying out the disjunctions. Rule (89) takes care of the simpler case when the disjunction names are identical. Rules (90), (91) and (92) on the other hand takes care of *inconsistency* propagation. The constraint $X : \bot$ is to be interpreted as a CL-ONE internal constraint which indicates that the variable X is inconsistent.

5.10 Negation in CL-ONE

Historically speaking the interpretation of negation has been a problematic issue within feature logics. The provision of negations within *feature structures* was first explored in [Karttunen 84] within the computational framework of D-PATR [Karttunen 86b]. *Karttunnen's* interpretation of negation is a non-monotonic one. Under this interpretation a description of the form $S \sqcap \neg T$ is satisfiable iff $[\![S]\!]^{\mathcal{I},\alpha} \cap [\![T]\!]^{\mathcal{I},\alpha} = \emptyset$ in every interpretation $\mathcal{I}$. Thus under this interpretation the feature description $(g : a) \sqcap \neg(g : a \;\&\; f : b)$ is inconsistent since there exists an interpretation that makes $[\![g : a]\!]^{\mathcal{I},\alpha} = [\![g : a \;\&\; f : b]\!]^{\mathcal{I},\alpha}$. However, its specialisation $(g : a \;\&\; f : a) \sqcap \neg(g : a \;\&\; f : b)$ is consistent since there is no interpretation $\mathcal{I}$ such that $[\![g : a \;\&\; f : b]\!]^{\mathcal{I},\alpha} \neq [\![g : a \;\&\; f : b]\!]^{\mathcal{I},\alpha}$. Hence, this definition of *negation* is not monotonic.

Yet another form of negation is *inequations* in Prolog-II [Colmerauer 86] which only permits inequality constraints between variables. Only descriptions which involve variables such as $X \sqcap \neg X$ can ever become inconsistent under this interpretation.

Another solution is to treat negations simply as a complementation operation in the semantics following [Smolka 88], as we have done for the language $\mathcal{ALS}$ (see 3). Under this interpretation the denotation of a negated term $\neg T$ is given by $\mathcal{U}^I - [\![T]\!]^I_{\alpha,\kappa}$. Within this approach the denotation of a negated feature term $f : T$ is provably equivalent to $f \uparrow \sqcup f : \neg T$. Similarly, the denotation of a conjunctive term such as $\neg(S \sqcap T)$ is equivalent to $\neg S \sqcup \neg T$. We shall call this form of negation *general negation.*

Although providing general negation is a desirable feature in any knowledge representation language, it also leads to gross computational inefficiency. This arises mainly because as we have seen conjunctive descriptions (which are the most widely used operation in practical applications) when negated give rise to disjunctions which are a major source of inefficiency in computational systems.

With these points in mind it was decided that the simpler and computationally efficient *inequations* of the form $X \sqcap \neg Y$ (*i.e.* general negations restricted to variables) would be provided in CL-ONE.

5.11 Type system

So far we have talked about various term forming constructs in CL-ONE incorporating various subsets of the language $\mathcal{ALO}$ and its extensions that we developed in Chapters 2 through 4. However for practical purposes the term language alone is insufficient unless there is a mechanism to maintain a database of term definitions. A definitional mechanism permits definitions of concept symbols and additionally relation symbols to be maintained. Such a mechanism would be analogous to *concept definitions* in knowledge representation languages such as KL-ONE [Brachman & Schmolze 85], CLASSIC [Bordiga *et al* 89], LOOM [MacGregor 88] and BACK [von Luck *et al* 87].

The definitional mechanism that we shall adopt for CL-ONE follows the approach taken in terminological languages in that a **concept definition** is a definition of the form:

$T = T_{def}$
where T is a CL-ONE sort symbol and T_{def} is a CL-ONE term and
the symbol $=$ is to be understood as non-commutative.

A CL-ONE **terminology** W is a collection of *concept definitions.* A terminology W is **acyclic** if the relation $=$ is well-founded with respect to the concept symbols.

Example : The following CL-ONE terminology defines the concepts *grandparent*, *parent*, *father*, *mother* etc.

(93)
$$grandparent = person \sqcap child : parent.$$
$$parent = person \sqcap child : person.$$
$$person = name : \top \& age : \top \& sex : \top.$$
$$male = sex : m.$$
$$female = sex : f.$$
$$mother = parent \sqcap female.$$
$$father = parent \sqcap male.$$

This specification states that a grandparent is a person whose child is a parent. Similarly, a parent is somebody whose child is a person.

Following [Carpenter *et al* 91] we shall interchangeably call a CL-ONE terminology a CL-ONE **type system**. Similarly, we shall interchangeably refer to *concept definitions* as **type definitions.** Furthermore, we shall use the term **types** to mean *defined sort symbols* while we shall use the term **sorts** to mean sort symbols that are *not* defined within a terminology.

We say that a relational algebra $\mathcal{I}$, an $\mathcal{I}$-assignment α and a context assignment κ satisfies a terminology W if:

- for every concept definition $T = T_{def} \in W$:
  - $[\![T]\!]^{\mathcal{I},\alpha,\kappa} = \alpha(T)$
  - $\alpha(T) = [\![T_{def}]\!]^{\mathcal{I},\alpha,\kappa}$

According to the above semantics, any interpretation that makes the interpretation of a type symbol equal to its definition counts as a valid interpretation. Following the work by [Nebel 90] we shall call the above semantics a *descriptive semantics* although we shall not be adopting this semantics for CL-ONE. Note that cyclic terminologies are accounted for by the above semantics.

An alternative method for providing semantics for terminologies is to follow the approach taken by [Höhfeld & Smolka 88]. Following their approach an alternative interpretation for terminologies can be provided as follows.

Let $\mathcal{I}$ be a relational algebra, κ be a context assignment and α_0 be a primitive assignment function that interprets:

1. every sort symbol C (*i.e.* undefined sort symbol) by a subset of $\mathcal{U}^I$ *i.e.* $\alpha_0(C) \subseteq \mathcal{U}^I$
2. every type symbol T by the empty set *i.e.* $\alpha_0(T) = \emptyset$

Then a **definite interpretation** of W with respect to $\mathcal{I}, \alpha_0, \kappa$ is an interpretation $\mathcal{I}, \alpha, \kappa$ defined iteratively as follows:

(94)
$$\alpha_0(T) = \emptyset$$
$$[\![T]\!]^{\mathcal{I},\alpha_{n-1},\kappa} = \alpha_{n-1}(T),\ n > 0$$
$$\alpha_n(T) = [\![T_{def}]\!]^{\mathcal{I},\alpha_{n-1},\kappa},\ n > 0$$
$$\alpha(T) = \bigcup_{i>0} [\![T_{def}]\!]^{\mathcal{I},\alpha_i,\kappa}$$

Again, cyclic terminologies present no problems for the definite interpretation, although it will often lead to "infinite objects" as models of a given type symbol. We shall call this semantics *definite semantics.*

The design decision adopted in CL-ONE is to adopt the definite semantics as defined above. This decision was taken mainly because of the fact that while there are *decidable* consistency checking algorithms known to work along with cyclic $\mathcal{ALC}$ terminologies using the descriptive semantics [Baader 91] we have not been able to investigate the effect of applying similar techniques to the languages $\mathcal{ALV}, \mathcal{ALS}$ and $\mathcal{ALO}$ (see [Nebel 91] and [Baader 90] for a comparison of the various different semantics). Nevertheless we do believe that such techniques are applicable to these logics and should be investigated in the future. On the other hand, there is a known characterisation of the *definite* semantics applicable to arbitrary constraint languages. [Höhfeld & Smolka 88] provide an operational and logical semantics of cyclic terminologies which are called *definite relations* (see section 5.12 for a correspondence) within a constraint logic programming setting. This means that the operational characterisation of consistency checking in cyclic CL-ONE terminologies comes for free. Of course, one disadvantage of this approach is that consistency checking of term descriptions involving cyclic terminologies is in general *undecidable.* Thus cyclic terminologies must be handled with care.

5.11.1 Multiple inheritance

One big advantage of the set denoting semantics of CL-ONE terms and terminologies is that providing *multiple inheritance* in CL-ONE is quite a simple task - at least from a theoretical perspective.

Example : The example given in (93) can be alternatively encoded as follows using inheritance statements.

(95)
$$grandparent = child : parent.$$
$$parent = child : person.$$
$$person = name : \top \& age : \top \& sex : \top.$$
$$male = sex : m.$$
$$female = sex : f.$$
$$mother = female.$$
$$father = male.$$

$$grandparent \leq person.$$
$$parent \leq person.$$
$$mother \leq parent.$$
$$father \leq parent.$$

One advantage of the above specification as opposed to the one given in (93) is that it makes explicit all the intended implicit inheritance relationships and provides an *object-oriented* structuring to the knowledge base.

The interpretation of type symbols provided in (94) needs to be modified to account for inheritance specifications.

Let H be a set of inheritance specifications.

Let $inherit(t)$ denote the set given by:

$$inherit(t) = \{s \mid t \leq s \in H\}$$

Let $\&inherit(t)$ denote the intersection (*i.e.* conjunction) of type/sort symbols in $inherit(t)$. Then, inheritance specifications are handled by modifying the way type definitions are interpreted. This is done as follows.

Every type definition:

$$T = T_{def}$$

is treated as if equivalent to:

$$T = T_{def} \ \& \ \&inherit(T)$$

This shows that inheritance specifications can be eliminated syntactically.

However, the interpretation of inheritance specifications in CL-ONE is identical to that for sort symbols provided in section 5.3. In other words, inheritance specifications are transparent to whether the given sort symbol is defined or not. This means that the issues of *closed world reasoning* are equally applicable to CL-ONE inheritance specifications.

This raises the following question : What form of *closed world reasoning* do we apply to CL-ONE inheritance specifications for types? Do we want to employ *glb semantics* to types as well as sorts?

Providing an identical and hence transparent semantics for inheritance specification for both types and sorts has a certain intuitive appeal. For instance, the language LOGIN [Aït-Kaci & Nasr 86] adopts this approach.

On the other hand, for efficiency reasons, one may opt for a *open-world semantics* for inheritance specification of CL-ONE types while choosing a *glb semantics* for CL-ONE sort symbols. This raises the following problem. How do we treat conjunctions of sort symbols and type symbols? In other words, given a *sort symbol* s and a type symbol t, do we interpret the expression $s \ \& \ t$ by $glb(s,t)$? A simple solution would be *not* to enforce this requirement and hence apply an open-world semantics to such cases.

Yet another solution would be to allow the user to choose between whether or not to apply a closed-world reasoning at all? This choice could be made available for interpreting both sorts and types.

Each of the above design alternatives can be considered a viable option for CL-ONE. And we shall leave it to a given implementation of CL-ONE to make a choice.

5.12 Relational dependencies

Relational dependencies such as the *append* relation are used in the HPSG grammar formalism to encode principles of grammar such as the subcategorisation principle.

A typical definition for *append* would be as given in (96).

(96)
$$list = nil.$$
$$list = h : top\&t : list.$$

$$append(nil, X\&list, X\&list).$$
$$append(h : X\&t : Y\&list, Z\&list, h : X\&t : W\&list) : -$$
$$append(Y, Z, W).$$

The framework of [Höhfeld & Smolka 88] adds *relational atoms* (*e.g. append*) to an arbitrary constraint language and provides both a denotational and operational semantics for the augmented language. Indeed our definite semantics for type definitions provided in section 5.11 is an adaptation of this semantics. Their operational semantics can be considered as a generalisation of SLD-resolution that is employed in conventional logic programming [Lloyd 84].

Although relational dependencies appear to be distinct from sorts as our previous discussion indicated they can easily be translated into sorts as the example in (97) shows.

(97)
$$\begin{aligned} append \;=\; & arg1 : nil \;\&\\ & arg2 : (X\&list) \;\&\\ & arg3 : (X\&list). \end{aligned}$$
$$\begin{aligned} append \;=\; & arg1 : (h : X \;\&\; t : (Y\&list)) \;\&\\ & arg2 : (Z\&list) \;\&\\ & arg3 : (h : X \;\&\; t : (W\&list)) : -append(Y, Z, W). \end{aligned}$$

This translation highlights a mechanism for showing certain equivalences between first-order terms and feature terms (*see* [Carpenter 91] and [Smolka & Treinen 92] for more on the subject).

For a CL-ONE implementation, the easiest route for integrating relational dependencies would simply be to provide a translation mechanism for syntactically translating relational dependencies with the above illustrated mechanism. This would have the benefit of a simple implementation and futhermore permit inheritance between relations.

On the other hand, handling relational dependencies separately does appear to have certain advantages as far as efficiency is concerned. Prolog implementations make crucial use of the fact that if a predicate head does not unify with the atom to be reduced then the clauses belonging to the predicate need not be added to the goal list. This often leads to a drastic reduction in the search space.

Within the framework of [Höhfeld & Smolka 88] this idea is generalised to what they dub as *V-constraint solving* which effectively allows a Prolog-like strategy to be employed for constraint languages. This means that a separate implementation of relational dependencies can take advantage of the implicit control information present in relational dependencies which would otherwise be lost if relational dependencies were eliminated in favour of type symbols.

Either option is a viable alternative for a CL-ONE implementation. A simple translation of relational dependencies to type symbols would be the choice in a first CL-ONE implementation and a separate mechanism for dealing with relational dependencies could be added later.

Finally to deal with disjunctive specifications in the definitional mechanism we assume that a specification such as:

$$t = t_{def_1}$$

$$\ldots$$

$$t = t_{def_n}$$

is interpreted as (if being) equivalent to:

$$t = t_{def_1} \sqcup \ldots \sqcup t_{def_n}$$

i.e. they are interpreted as the union.

Operationally speaking we shall assume the existence of a Prolog style backtracking strategy for dealing with disjunctive specifications.

5.13 Relation typing

While type definitions restrict the class of structures that are admitted as models of a given concept, **relation typing** achieves the same effect for relation symbols.

A simple typing scheme on relation symbols would permit the specification of the *domain* and *range* of the relation symbol. For instance, the relation symbol **sex** can be thought of as a function that relates a concept of type **person** (its domain) to a concept of type **sex** (its range) (- note the two distinct uses of the symbol *sex*). Thus a simple typing definition for the relation *sex* would look like:

(98) $sex : person \rightharpoonup sex$

This definition indicates that the relation symbol *sex* is to be interpreted as a partial function that relates a *person* to its *sex*.

On the other hand, the relation symbol *child* that relates a *person* to its *child* can be specified as follows:

(99) $child : person \rightharpoonup 2^{person}$

This definition states that the relation *child* relates a *person* to its *children*. However, it permits a person to have more than one child.

Relation typing is a common mechanism for specifying the upper bound on the domain and range of relation symbols in concept languages - *e.g.* the language LOOM [MacGregor & Bates 87] and BACK [von Luck *et al* 87] support such definitions.

Relation typing achieves roughly the same effect for relations what type definitions achieve for concept symbols.

We shall assume that relation typing definitions are part of a terminology. This means that every interpretation that satisfies a given terminology W will have to respect all the relation typing definitions specified in W.

There are at least two different ways in which relation typing specifications can to be interpreted. A simple interpretation of relation typing specification would be the obvious one given in (100).

Notation: In the following and throughout this Chapter we shall use $2^{[t]^{\mathcal{I},\alpha,\kappa}}$ to denote the **non-empty** subsets of $[\![t]\!]^{\mathcal{I},\alpha,\kappa}$.

(100) **Strong Partial Typing**

$$\mathcal{I},\alpha,\kappa \models f : s \rightharpoonup t \Longleftrightarrow f^I : [\![s]\!]^{\mathcal{I},\alpha,\kappa} \rightharpoonup [\![t]\!]^{\mathcal{I},\alpha,\kappa}$$
$$\mathcal{I},\alpha,\kappa \models f : s \rightharpoonup 2^t \Longleftrightarrow f^I : [\![s]\!]^{\mathcal{I},\alpha,\kappa} \rightharpoonup 2^{[t]^{\mathcal{I},\alpha,\kappa}}$$

Thus according to this definition, a relation typing definition is a partial function from it domain to (the powerset of) its range. This definition can be considered as a *strong* interpretation of relation typing specification since it assumes that there the relation typing definition provides complete information on the domain and range of the relation.

A weaker interpretation of relation typing specification can be provided by assuming that the typing information is only a partial description of the domain and range of the relation. This can be achieved as follows.

Let $f \restriction^{\phi}$ denote the function obtained by restricting f to the members of the set ϕ.

Then a relation typing specification can be alternatively interpreted by the following definitions.

(101) **Weak Partial Typing**

$$\mathcal{I},\alpha,\kappa \models f \restriction: s \rightharpoonup t \Longleftrightarrow f^I \restriction^{[s]^{\mathcal{I},\alpha,\kappa}}: [\![s]\!]^{\mathcal{I},\alpha,\kappa} \rightharpoonup [\![t]\!]^{\mathcal{I},\alpha,\kappa}$$
$$\mathcal{I},\alpha,\kappa \models f \restriction: s \rightharpoonup 2^t \Longleftrightarrow f^I \restriction^{[s]^{\mathcal{I},\alpha,\kappa}}: [\![s]\!]^{\mathcal{I},\alpha,\kappa} \rightharpoonup 2^{[t]^{\mathcal{I},\alpha,\kappa}}$$

Note that we have chosen a different syntax to represent the different usage of the relation typing definition.

According to the above definition, a relation typing specification such as *sex* $\restriction$:

person $\rightharpoonup$ *sex* states that the relation *sex* (partially) relates a *person* to its *sex* but is agnostic about the way the relation *sex* relates concepts other than *persons*.

While the above considerations provide one dimension of variation on the interpretation of relation typing definitions, an additional mechanism would be to permit relation typing specification to be total functions as opposed to partial functions. For this purpose we shall use the following syntax and semantics given in (102).

(102) **Strong Total Typing**

$$\mathcal{I}, \alpha, \kappa \models f : s \longrightarrow t \Longleftrightarrow f^I : [\![s]\!]^{\mathcal{I},\alpha,\kappa} \longrightarrow [\![s]\!]^{\mathcal{I},\alpha,\kappa}$$
$$\mathcal{I}, \alpha, \kappa \models f : s \longrightarrow 2^t \Longleftrightarrow f^I : [\![s]\!]^{\mathcal{I},\alpha,\kappa} \longrightarrow 2^{[\![s]\!]^{\mathcal{I},\alpha,\kappa}}$$

(103) **Weak Total Typing**

$$\mathcal{I}, \alpha, \kappa \models f \restriction: s \longrightarrow 2 \Longleftrightarrow f^I \restriction^{[\![s]\!]^{\mathcal{I},\alpha,\kappa}} : [\![s]\!]^{\mathcal{I},\alpha,\kappa} \longrightarrow [\![t]\!]^{\mathcal{I},\alpha,\kappa}$$
$$\mathcal{I}, \alpha, \kappa \models f \restriction: s \longrightarrow 2^t \Longleftrightarrow f^I \restriction^{[\![s]\!]^{\mathcal{I},\alpha,\kappa}} : [\![s]\!]^{\mathcal{I},\alpha,\kappa} \longrightarrow 2^{[\![t]\!]^{\mathcal{I},\alpha,\kappa}}$$

We shall call the above typing definitions **total typing** while we shall dub the one given in (100) and (101) as **partial typing**.

According to the above definitions, a relation typing specification such as:

$$sex : person \longrightarrow sex$$

ensures that the *sex* of every *person* must be specified.

Under the total typing scheme, relation typing definitions can easily lead to infinite models. For instance, the specification $father : person \longrightarrow person$ requires that every *person* must have a *father* who is a *person* and hence must also have a *father* and so on - something that cannot be accommodated within the finite memory of a computer (*unless* we permit cyclic objects as models).

One way to bring determinism to consistency checking with a total relation typing specification is to insist on an *acyclicity condition* on the chain of relation typing definitions *i.e.* require that the relation $\longrightarrow^*$ (which denotes the transitive closure of the total typing relation $\longrightarrow$) is acyclic. This is similar to the approach taken in the design of the type system for the ALE formalism [Carpenter 93].

An alternative would be not to insist on any restrictions but ensure that the user is aware of non-termination (which is in any case present due to cyclic type definitions) if there are cycles *w.r.t* a total relation typing specification.

Finally, the four different schemes of relation typing we presented in this section can co-exist and a given implementation can provide separate syntax to specify all the different types of relation typing specifications allowing the user to choose whichever scheme is most appropriate for the application at hand.

5.13.1 Consistency checking with relation typing specifications

In this section we present consistency checking rules that augment the consistency machinery of the language CL-ONE with rules to handle relation typing specifications.

(DomRanApp) $\dfrac{C_s}{\{x : s,\ y : t\} \cup C_s}$

if $x \ \dot{f} \ y \in C_s$ and the following conditions hold:

1. either: (a) $f : s \rightharpoonup t \in W$, (b) $f : s \rightharpoonup 2^t \in W$,
   (c) $f : s \longrightarrow t \in W$ or (d) $f : s \longrightarrow 2^t \in W$

   and
2. it is not the case that

   $\exists x : s', y : t' \in C_s$ such that $s' \leq s$ and $t' \leq t$

(RanApp) $\dfrac{\{x : s'\} \cup C_s}{\{x : s',\ y : t\} \cup C_s}$

if $x \ \dot{f} \ y \in C_s$ and the following conditions hold :

1. either : (a) $f \upharpoonright : s \rightharpoonup t \in W$ or
   (b) $f \upharpoonright : s \longrightarrow t \in W$

   such that $s' \leq s$ and
2. $\nexists y : t' \in C_s$ such that $t' \leq t$

(FeatApp) $\dfrac{\{x : s'\} \cup C_s}{\{x : s',\ xfy\} \cup C_s}$

if the following conditions hold:

1. y is new
2. $x \ \dot{f} \ y' \notin C_s$ and
3. either : (a) $f : s \longrightarrow t \in W$ or
   (b) $f \upharpoonright : s \longrightarrow t \in W$

   such that $s' \leq s$

(RelApp) $\dfrac{\{x : s'\} \cup C_s}{\{x : s',\ x\ \exists f\ y,\ y : t\} \cup C_s}$

if the following conditions hold:

1. y is new
2. $x \ \dot{f} \ y' \notin C_s$ and
3. either : (a) $f : s \longrightarrow 2^t \in W$ or
   (b) $f \upharpoonright : s \longrightarrow 2^t \in W$

   such that $s' \leq s$

Figure 5.16: **Relation typing rules - I**

Notation:

1. We shall write $x \ \dot{f} \ y_i \in C_s$ if either of the following conditions hold:
   - $xfy_i \in C_s$
   - $x \ \exists f \ y_i \in C_s$
   - $x = f : \{\ldots, y_i, \ldots\} \in C_s$
2. We shall write $x \ \dot{f} \ y_i \notin C_s$ if there are no variables x, y_i such that $x \ \dot{f} \ y_i \in C_s$

We illustrate the basic idea with our relation typing rules by the following rule.

(DomRanApp) $\dfrac{C_s}{\{x : s, \ y : t\} \cup C_s}$

if $x \ \dot{f} \ y \in C_s$ and the following conditions hold:

either: (a) $f : s \rightharpoonup t \in W$, (b) $f : s \rightharpoonup 2^t \in W$,
(c) $f : s \longrightarrow t \in W$ or (d) $f : s \longrightarrow 2^t \in W$

The rule DomRanApp states that if for instance, we have $x \ sex \ y \in C_s$ and supposing we have $sex : person \longrightarrow sex \in W$ then we need to add $x : person$ and $y : sex$ to C_s. A similar argument will apply if instead of $sex : person \longrightarrow sex \in W$ we have any of $f : s \rightharpoonup 2^t \in W$, $f : s \longrightarrow t \in W$ or $f : s \longrightarrow 2^t \in W$.

As such the rule given above will apply indefinitely, so we need to add some checks to ensure that the rule has not previously applied.

The complete set of rules is presented in figures 5.16 and 5.17.

The rules given in 5.17 are intended to deal with relation typing definitions that specify that a given relation symbol is functional as opposed to being relational. For instance, the definition $sex : person \longrightarrow sex \in W$ states that the relation symbol sex is to be interpreted as a total function from $person$ to sex. Assuming that $male \leq sex \in W$ and $female \leq sex \in W$ then $x \ \exists sex \ y, y : male, x \ \exists sex \ z, z : male \in C_s$ should result in an inconsistency. The rules given in figure 5.17 capture this kind of behaviour. It is fairly easy to see that our rules are invariant and complete. However, termination cannot be guaranteed in the face of cyclic relation typing specifications.

$$\text{(RelFun)} \quad \frac{\{x\ \exists f\ y\} \cup C_s}{\{xfy\} \cup C_s} \text{ if } f : s \longrightarrow t \in W$$

$$\text{(SetFun)} \quad \frac{\{x = f : \{y_1, \ldots, y_n\}\} \cup C_s}{\{xfy_1,\ y_1 = y_2,\ \ldots\ ,\ y_1 = y_n\} \cup C_s} \text{ if } f : s \longrightarrow t \in W$$

$$\text{(DomRelFun)} \quad \frac{\{x : s',\ x\ \exists f\ y\} \cup C_s}{\{x : s',\ xfy\} \cup C_s} \text{ if } f \restriction : s \longrightarrow t \in W \text{ and } s' \leq s$$

$$\text{(DomSetFun)} \quad \frac{\{x : s',\ x = f : \{y_1, \ldots, y_n\}\} \cup C_s}{\{x : s',\ xfy_1,\ y_1 = y_2,\ \ldots\ ,\ y_1 = y_n\} \cup C_s}$$
$$\text{if } f \restriction : s \longrightarrow t \in W \text{ and } s' \leq s$$

Figure 5.17: **Relation typing rules - II**

5.14 Implementation

Although a complete prototype of CL-ONE has not been implemented, various modules of CL-ONE have been successfully implemented. These include the boolean constraint solver in conjunction with the partially ordered sorts obeying the *glb* semantics, the module for dealing with sets and the module for dealing with linear precedence constraints.

The biggest hurdle for a realistic implementation proved to be the difficulty in attaching and detaching constraints on a variable using the currently available Prolog technology. Instead very large Prolog datastructures had to be continuously updated as the constraint solver considered new constraints. In addition our abstract constraint solving algorithm for dealing with linear precedence constraints is not very efficient since it introduces a large number of new constraints. This quickly leads to memory requirements beyond the capacity of the machine. On the other hand, the nature of the project prevented a low level implementation.

However we believe that with further work on efficient datastructures for dealing with linear precedence constraints and with the aid of new control primitives [Holzbaur 90] that are becoming available in newer versions of Prolog a realistic implementation is possible.

5.15 Summary

In this Chapter we have provided a concrete design study for a constraint language that significantly extends currently implemented formalisms by providing direct support for constraints involving set descriptions and linear precedence constraints. The language CL-ONE can be considered expressive enough to model a significant part if not all of the HPSG grammar formalism. In addition to these devices CL-ONE also provides specialised support for restricted distributed disjunctions and restricted boolean sort constraints which are aimed primarily at providing very efficient representation to deal with a restricted class of disjunctions and propositional sort constraints. Our boolean constraints can be considered to be a restricted version of that found in the STUF formalism [Dörre & Seiffert 91] but which we believe to be sufficient for most linguistic purposes. We believe that these additions go a long way in achieving sufficient linguistic coverage.

For an implementation of CL-ONE to be successful, there is a fair amount of work that needs to be done. Firstly, we have not studied efficient algorithms for dealing with constraints involving set descriptions. A successful implementation of CL-ONE should at least provide a reasonably efficient method for dealing with set-membership constraints since these we believe are the most widely needed form of constraints involving set descriptions. Finally, work needs to be done on investigating efficient techniques of implementing linear precedence constraints since the computation method that is suggested by our rewrite rules is clearly not computationally very efficient.

On the logical side, our current formulation of the language $\mathcal{ALS}$ does not provide a *union* operation on set descriptions. This device would be useful for instance for a "set-based" version of the *subcategorisation principle* which uses set-valued subcategorisation frames instead of lists as is the case with the current formulation of HPSG [Pollard & Sag 92].

Similarly, if set-union constraints are available then our linear precedence constraints could be formulated in an alternative manner with a much simplified constraint solving machinery.

To address the above limitation, in Appendix A we extend a restricted sublanguage of $\mathcal{ALS}$ with the *subset, union, intersection* and *disjoint union* operations. However further work is essential to verify the completeness and termination properties of this extended logic.

Chapter 6

Linguistic Applications

In this Chapter we describe some potential applications of a formalism such as CL-ONE for representing linguistic knowledge. The applications covered in this Chapter exclude the application of CL-ONE for describing word-order which have already been covered in Chapter 4. Our description is not intended as an in-depth survey of the linguistic applicability of CL-ONE but rather as a pointer into how the expressive potential of such a formalism might be exploited.

The applications we explore are mainly to do with the specification of thematic role based argument selection principles, the declarative formulation of subject selection principles based on thematic *proto-roles* [Dowty 88] and a sign based formulation of DRT [Kamp 81].

We demonstrate how the theories of argument selection based on thematic marking of arguments can be captured as high level CL-ONE specifications taking advantage of the constraints on set descriptions.

We develop a computational formulation of the theory of thematic proto-roles showing that the availability of finite domain constraints in CL-ONE provides a clean high level specification of Dowty's subject selection principles [Dowty 88].

The approach taken in this Chapter is intended to provide a starting point for declarative specification of linguistic theories that assume a central role for the thematic content of lexical items.

Finally we provide a reformulation of the Prolog based logical specification of DRT developed in [Johnson & Klein 86] within CL-ONE augmented with the set union construct. This shows how the theory of DRT can be embedded within HPSG. We also believe that our formulation of DRT is at a higher level than that provided in [Johnson & Klein 86].

6.1 Thematic roles and argument selection

The theory of thematic roles is largely motivated by the fact that syntactic properties of lexical items such as argument selection and preposition selection can be predicted on the basis of their thematic properties. This essentially defines a partial mapping between *thematic grids* and the *grammatical function* of the thematic arguments. In this section we shall take insights from existing theories of thematic roles and illustrate how this mapping can be stated as CL-ONE definitions.

Within a role-based approach to grammar, lexical entries only specify the thematic roles of each argument instead of having to fully specify their syntactic properties. For instance, the entry for the verb *give* would look like:

$$give < agent, theme, goal >$$

For the verb *hit* there would be two entries:

$$hit < instrument, goal >$$

and

$$hit < agent, goal >$$

A rough definition of the intended meaning of each thematic role can be provided as follows:

THEMATIC ROLE	FUNCTION
agent	A sentient being responsible for the action
theme	The object that caused the action to happen
goal	The entity towards which something moves
instrument	The artifact responsible for bringing about the event
source	The place where the event was initiated
destination	The place where the event ends
locative	The location where the event took place

6.1.1 Argument Selection Principles

Fillmore in [Fillmore 68] suggests that the syntactic realisation of an argument as a NP or a PP can be predicted on the basis of thematic roles.

Consider the following examples:

(104) a. Harry sprayed *paint* [on the wall].

b. Harry sprayed [the wall] *with paint.*

c. *Harry sprayed [on the wall] *with paint.*

d. *Harry sprayed [the wall] paint.

(105) a. John planted *peas and corn* [in his garden].

b. John planted [his garden] *with peas and corn.*

c. *John planted [in his garden] *with peas and corn.*

d. *John planted [his garden] *peas and corn.*

(106) a. I loaded *hay* [on the truck].

b. I loaded [the truck] *with hay.*

c. *I loaded [on the truck] *with hay.*

d. *I loaded [the truck] *hay.*

(107) a. I smeared *mud* [on the wall].

b. I smeared [the wall] *with mud.*

c. *I smeared [on the wall] *with mud.*

d. *I smeared [the wall] *mud.*

A plausible analysis of the above data is that in each case the *italicised* expression is an INSTRUMENT, and the bracketed expression is a LOCATIVE (see [Radford 88] *pp.* 380). In other words, the lexical entry for the above class of verbs will simply contain the thematic grid:

$$< Agent, Instrument, Locative >$$

Fillmore suggests that general argument selection principles in the above class of verbs would imply that only one non-subject argument may be realised as a NP while the other non-subject argument has to be realised as a PP.

Yet another approach to argument selection principles is the work of [Williams 81] who suggests a general set of rules that govern the syntactic realisation of the non-subject *theme* and *goal* arguments. The following rules are suggested in [Williams 81]:

(i) Theme: (NP)

(ii) Goal: $(\underline{NP}, PP_{to})$

(iii) Goal: (NP_2)

Following Williams's notation The first rule (i) states that the *theme* argument of a predicate is always realised as a bare NP. The second rule (ii) states that the *goal* argument of a predicate can be realised as a NP headed by the preposition *to*. The notation $(\underline{NP}, PP_{to})$ is intended to state that PP_{to} is the mother node of a tree whose only daughter is $\underline{NP}$. The third rule states that the *goal* argument can also be realised as a bare NP in a *direct object* (indicated by the subscript 2).

With this added set of principles we can explain the behaviour of verbs such as *give*:

$$give < Agent, Theme, Goal >$$

in the following examples.

(108) a. John gave *Mary* a book.

b. John gave a book *to Mary*.

c. ?John gave *to Mary* a book.

Since the goal argument *to Mary* functions as the direct object in sentence (a), it may be realised as a bare NP following rule (iii). On the other hand, in sentences (b) and (c) the goal argument is realised as a PP and is licenced by rule (ii). The markedness of the sentence (c) can be captured from independent word-ordering principles which state that NP arguments should precede their PP counterparts.

6.1.2 Preposition selection

Argument selection principles as discussed above force an argument to be realised as either as a NP or a PP but in the case a PP is realised the general principles do not state which preposition needs to be selected. This raises the further question as to how prepositions are to be correctly predicted. Are there some general rules to determine the distribution of prepositions in non-subject arguments? Or, does each lexical entry randomly determines its prepositional arguments? Note that in English *Subjects* are usually realised as bare NPs so the question is only valid for internal arguments.

Fortunately, it turns out that to a large degree it is possible to determine the correct preposition from the thematic role of the PP argument. This means that in a large number of cases it is possible to (at least partially) eliminate *prepositional* markings of thematic arguments.

Fillmore showed that one of the functions of prepositions is to theta-mark (*i.e.* assign a theta role to) their argument NPs. Of course, we do not assume that there is a one-to-one correspondence between prepositions and the theta-roles they assign. There could be the possibility that a given preposition assigns different thematic roles. As a rough guide to the set of prepositions and their thematic function we shall use the following table modified from [Radford 88]:

PREPOSITION	θ-FUNCTION	PREPOSITION	θ-FUNCTION
in	LOCATIVE	*on*	LOCATIVE
onto	LOCATIVE	*to*	GOAL or DESTINATION
by	AGENT	*with*	INSTRUMENT
from	SOURCE	*for*	BENEFACTIVE

This shows that not only arguments that are realised as PPs can be predicted but the preposition can also be predicted based on the thematic compatibility between the preposition and the argument thematic role.

Of course cases remain such as verbs *blame, accuse, charge* that idiosyncratically select the prepositions *for, of, with* respectively (see [Pollard and Sag 88] *pp.* 127).

1. The authorities blamed Greenpeace for/*with/*of the bombing.
2. The authorities accused Greenpeace *for/*with/of the bombing.
3. The authorities charged Greenpeace *for/with/*of the bombing.

For such idiosyncratic verbs a rough and ready treatment is available by employing the PFORM feature which identifies a specific preposition.

6.1.3 Encoding argument and preposition selection principles in CL-ONE

In this section we show mainly via examples how thematic information in lexical entries can be encoded as CL-ONE descriptions.

We assume that both NPs and PPs are specified as $\left[\text{HEAD|MAJ } N\right]$. We shall then distinguish NPs from prepositions by having the value of the path ROLE|BARE as *plus* where prepositions will have the value *minus*. In other words, we shall assume the following definitions.

(109) $np = \left[\text{ROLE|BARE } plus\right]$

(110) $pp = \left[\text{ROLE|BARE } minus\right]$

Arguments which can be realised either as a PP or a NP have an *unspecified* value for the path ROLE|BARE. On the other hand verbal arguments that need to be realised as a bare NP will have *plus* instantiated as the value of the path ROLE|BARE.

For the purposes of this Chapter, we shall assume that the grammatical function of an argument is specified as the value of the path ROLE|SYN while the thematic role of an

argument will be specified as the value of the path ROLE|THETA. We shall also assume that at least the grammatical functions *sub* (standing for *subject*) and *do* (standing for *direct object*) need to be determined by syntactic realisation principles.

As an example, the simplified lexical entry for the verb *give* will contain at least the following information:

(111) $$\left[\text{SYNSEM|SYN|ARGS}\left\{\left[\text{ROLE|THETA } \textit{agent}\right],\left[\text{ROLE|THETA } \textit{goal}\right],\left[\text{ROLE|THETA } \textit{theme}\right]\right\}\right]$$

For example an entry for the preposition *to* would be defined as follows.

(112) $$\textit{preposition} = \left[\text{SYN}\left[\text{ARGS}\left\{\textit{np}\right\}\right]\right] \;\&\; \textit{pp}$$

(113) $$\textit{to} = \left[\text{ROLE|THETA } \textit{goal} \sqcup \textit{destination}\right]$$
$$\textit{to} \leq \textit{preposition}$$

The entries for other prepositions can be specified in a similar manner.

Two points are in order here. Firstly, a preposition selects a NP (*i.e.* ROLE|BARE *plus*) as its argument. Secondly, the preposition assigns the appropriate theta-role to the whole PP phrase. In the above case the preposition *to* assigns a *goal* or *destination* role.

To encode argument selection principles as outlined in section 6.1.1 we proceed as follows. Firstly, to encode Williams' principle that the theme argument is always realised as a bare NP we use the following CL-ONE description:

(114) $$\textit{theme\_select} = \left[\text{SYN|ARGS } \forall : \left[\text{ROLE}\left[\begin{array}{ll}\text{THETA} & D(\textit{theme}, \neg\textit{theme}) \\ \text{BARE} & D(\textit{plus}, \top)\end{array}\right]\right]\right]$$

This entry states that every theme argument has to be realised as a NP while a non-theme argument can be realised as a either as a NP or a PP.

Secondly, to encode Williams' principle that the goal argument can be realised as bare NP in direct object position only we shall use the following CL-ONE description:

(115)
$$goal\_select = \left[\text{SYN|ARGS } \forall : \left[\text{ROLE} \left[\begin{array}{ll}\text{SYN} & D(do, \neg do, \top) \\ \text{THETA} & D(goal, goal, \neg goal) \\ \text{BARE} & D(plus, minus, \top)\end{array}\right]\right]\right]$$

This enforces the condition that if the goal argument is bare then it must be in the direct object position otherwise the goal argument has to be a PP.

Assuming that *incremental-themes* [Dowty 88] are realised as bare NPs just like *themes*, we can extend the CL-ONE description for *themes* to *incremental-themes*.

We shall assume that locative arguments are always realised as PPs and instrumental arguments are realised as a PPs in non-subject positions. However, an instrumental argument can be realised as a bare NP in the subject position as the following construction demonstrates:

(116) The rock broke the vase.

where the NP, *the rock*, functions as the instrument.

The following definitions take care of the syntactic realisation of these thematic arguments.

(117)
$$locative\_select = \left[\text{SYN|ARGS } \forall : \left[\text{ROLE} \left[\begin{array}{ll}\text{THETA} & D(locative, \neg locative) \\ \text{BARE} & D(minus, \top)\end{array}\right]\right]\right]$$

(118)
$$instrument\_select = \left[\text{SYN|ARGS } \forall : \left[\text{ROLE} \left[\begin{array}{ll}\text{SYN} & D(sub, \neg sub, \top) \\ \text{THETA} & D(instrument, instrument, \neg instrument) \\ \text{BARE} & D(plus, minus, \top)\end{array}\right]\right]\right]$$

To enforce the argument selection principles we have defined so far, we would need a definition such as the one given below.

(119) $arg\_selection\_principles =$ *theme_select* & *goal_select* & *inctheme_select* & *locative_select* & *instrument_select* ...

(120) *main_verbs = arg_selection_principles & ...*

With the argument selection principles in place, lexical entries can underspecify the syntactic properties of their arguments if these can be predicted from general thematic information.

6.2 Proto-roles and Subject selection

6.2.1 The theory of Proto-roles

Dowty [Dowty 88] bases his theory of thematic roles on what he calls *prototypical roles* or *proto-roles*. He proposes that languages to a large degree distinguish only the *proto-agent* and the *proto-patient* roles. Then it is the *proto-agent* role that gets realised as a *subject* (*i.e* an *external* argument) in a declarative sentence and as an *oblique* in a *passive* construction. How does the grammar then find out which role is going to effectively function as the *proto-agent*? Dowty claims that the notion of *proto-agent* (and likewise *proto-patient*) is not a single isolated property but its a cluster of related properties that are entailed by the semantics of both the verb and its arguments. It is possible for more than one argument position to have common contributing properties of being a *proto-agent* (likewise *proto-patient*). In such cases the argument that has the most contributing properties of the *proto-agent role* (likewise the *proto-patient* role) will be selected. In other words, arguments have to compete with each other for proto-agenthood.

One advantage of this approach is that it easily explains the behaviour of *symmetric predicates* such as *as tall as, as big as etc.* where either argument role can effectively function as the *proto-agent*.

(121) John is as tall as Mary.

(122) Mary is as tall as John.

What are the contributing properties of proto-agent and proto-patient roles?

Dowty has suggested that the argument which as compared with other arguments entails the greatest number of the following properties will be grammatically realised as the *proto-agent* [Dowty 88].

1. volition
2. sentience
3. causes event
4. movement
5. referent exists independent of action of verb

Similarly, the argument that entails the greatest number of the following properties will be realised as the *proto-patient.*

1. change of state (including coming to being, going, out of being)
2. incremental theme (*i.e.* determinant of aspect)
3. causally affected by event
4. stationary (relative to the motion of Proto-agent)
5. referent may not exist independent of the action of the verb, or may not exist at all

Proto-Agent & Proto-Patient Selection

The following *Argument Selection Principle* stated in [Dowty 88] illustrates the way in which *Subject Selection* and *Object Selection* take place under the theory of proto-roles:

> *Argument Selection Principle : The argument of a predicate having the greatest number of Proto-Agent properties entailed by the meaning of the predicate will, all things being equal, be lexicalised as the subject of the predicate, the argument having the greatest number of Proto-Patient properties will, all else being equal, be lexicalised as the direct object of the predicate.*
>
> *Corollary 1: If two arguments of a relation have (approximately) equal numbers of proto-agent and proto-patient properties, then either may be lexicalised as the subject (and similarly for objects).*

6.2.2 Proto properties of traditional thematic roles

What properties do traditional thematic roles have?

For instance, traditional AGENT roles have the property of being *volitional, sentient/perception, causes event* and usually causes *movement.* On the other hand, traditional THEME roles usually have the property of being *causally affected* and *stationary relative to the proto-agent.* However these traditional THEME roles may also undergo a *change of state* and *may not exist at all.* Thus one approach to a computational treatment of traditional roles would be to characterise them by the properties they entail.

As a rough guide the following table can be used for determining the proto-agent properties entailed by traditional roles:

P-Agent Prop	agent	patient	theme	instr.	source	dest.	inc_th.
VOLITION	1	1	0	0	0	0	0
SENTIENCE	1	1	0	0	0	0	0
CAUSE_EVNT	1	0	0	1	0	0	0
MOVEMENT	1			1	0	0	0
REF_EXIST			1	1	1	1	1

The values 1 or 0 indicate whether a given property is definitely entailed or not entailed.

A blank entry in the above table indicates that the value is underspecified for either 1 or 0.

Similarly the following table provides a rough guide for determining the proto-patient:

P-Patient Prop	agent	patient	theme	instr.	source	dest.	inc_th.
CH_STATE	0	1	0	0	0	0	0
INC_THEME	0	0	0	0	0	0	1
CSLY_AFTD	0	1	1	1	1	1	1
STATIONARY	0	1	0	0	1	1	0
REF_NOT_EXIST	0			0	0	0	

In addition to the above entries we shall assume that for propositional arguments whose thematic role is marked as *proposition* the value of every proto-agent and proto-patient property will be 0. This means that propositional arguments are least desirable for proto-agent or proto-patient selection.

6.2.3 Encoding Thematic Proto-roles in CL-ONE

The idea is to continue to use the traditional roles but now these roles will act as *definitions* that will be expanded according to the above table. For example, the role *agent* will be a shorthand for:

(123)
$$agent = \begin{bmatrix} \text{NAME} & agent' \\ \text{VOLITION} & 1 \\ \text{SENTIENCE} & 1 \\ \text{CAUSE\_EVNT} & 1 \\ \text{MOVEMENT} & 1 \\ \text{REF\_EXIST} & 1 \sqcup 0 \\ \text{CH\_STATE} & 0 \\ \text{INC\_THEME} & 0 \\ \text{CSLY\_AFTD} & 0 \\ \text{STATIONARY} & 0 \\ \text{REF\_NOT\_EXIST} & 0 \end{bmatrix}$$

Thus every property (such as VOLITIONAL) is associated with a corresponding feature label.

None of our previous lexical entries that used traditional roles need changing except that now they are no longer viewed as atomic symbols but as a cluster of contributing properties.

However, since our argument realisation principles still make use of traditional roles these need to modified slightly so that the path ROLE|THETA is changed to ROLE|THETA|NAME and the theta role such as *agent* has to replaced by *agent'*. Thus the principle that the goal argument can be realised as a bare NP in the direct object position only has to be changed to the following CL-ONE description.

(124)
$$goal\_select = \begin{bmatrix} \text{SYN|ARGS} \; \forall : \begin{bmatrix} \text{ROLE} & \begin{bmatrix} \text{THETA|NAME} & D(goal', goal', \neg goal') \\ \text{SYN} & D(do, \top, \top) \\ \text{BARE} & D(plus, minus, \top) \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

One benefit would be that arguments can be distinguished by whether they are *animate* or *inanimate* since these notions can be reduced to the above properties too as follows:

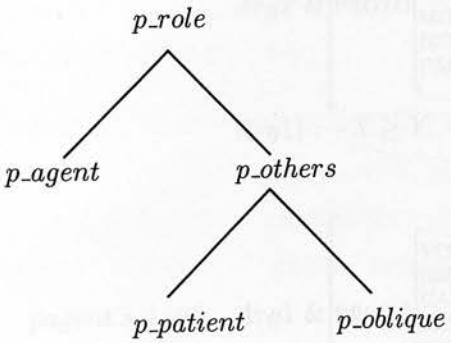
(125)
$$animate = \begin{bmatrix} \text{VOLITION} & 1 \\ \text{SENTIENCE} & 1 \end{bmatrix}.$$

(126) $$inanimate = \begin{bmatrix} \text{VOLITION} & 0 \\ \text{SENTIENCE} & 0 \end{bmatrix}.$$

Thus *nouns* including Wh-words can be further classified by their animacy. And, these semantic properties will provide further *selectional restrictions* whenever there is ambiguity in argument attachment.

Given the semantic notion of traditional thematic roles the job of grammar is then to use this knowledge to decide which role is going to function as the proto-agent. We shall achieve this by using a relational dependency *select_p_agent*. The job of this predicate is to select one of the supplied arguments as the proto-agent and instantiate the value of the path ROLE|PROTO to the atomic sort *p_agent* (thus identifying the proto-agent role). It will also need to instantiate the remaining arguments to the sort *p_others*. The idea is that *p_others* can be further split apart as $p\_others = p\_patient \sqcup p\_oblique$ which can be specified using a CL-ONE inheritance hierarchy as follows:

(127)



We shall assume that every *ditransitive* (and similarly every *transitive* verb) will inherit the following specifications.

(128) $$intrans\_p\_agent = \left[\text{SYN|ARGS } \exists : \left[\text{ROLE|PROTO } p\_agent\right]\right]$$

(129) $$intrans = intrans\_p\_agent \ \& \ \ldots$$

(130) $$trans\_p\_agent = \left[\text{SYN|ARGS}\left\{\boxed{1},\boxed{2}\right\}\right] : - \\ pagent\_select\left(\boxed{1},\boxed{2},\left[\text{ROLE|PROTO|NAME } p\_agent\right]\right).$$

(131) $$trans = trans\_p\_agent \ \& \ \ldots$$

(132) $$ditrans\_p\_agent = \left[\text{SYN|ARGS}\{\boxed{1},\boxed{2},\boxed{3}\}\right] :- \\ pagent\_select(\boxed{1},\boxed{2},\boxed{4}), \\ pagent\_select(\boxed{3},\boxed{4},\left[\text{ROLE|PROTO|NAME } p\_agent\right]).$$

(133) $$ditrans = ditrans\_p\_agent \;\&\; \ldots$$

The obvious way *pagent_select* is going work is by adding all the 1 and 0's (relevant to the proto-agent role) from each role entry and identifying the argument which has the largest possible value.

(134)
$$pagent\_select(\; Arg1 \;\&\; \left[\text{PROTO}\left[\begin{array}{ll}\text{VOLITION} & A1 \;\&\; 0..1\\ \text{SENTIENCE} & B1 \;\&\; 0..1\\ \text{CAUSE\_EVNT} & C1 \;\&\; 0..1\\ \text{MOVEMENT} & D1 \;\&\; 0..1\\ \text{REF\_EXISTS} & E1 \;\&\; 0..1\\ \text{PAGENT\_SUM} & (A1+B1+C1+D1+E1) \;\&\; X\end{array}\right]\right], \\ Arg2 \;\&\; \left[\text{PROTO}\left[\begin{array}{ll}\text{VOLITION} & A2 \;\&\; 0..1\\ \text{SENTIENCE} & B2 \;\&\; 0..1\\ \text{CAUSE\_EVNT} & C2 \;\&\; 0..1\\ \text{MOVEMENT} & D2 \;\&\; 0..1\\ \text{REF\_EXISTS} & E2 \;\&\; 0..1\\ \text{PAGENT\_SUM} & (A2+B2+C2+D2+E2) \;\&\; Y\end{array}\right]\right], \\ Arg1) : -X \geq Y.$$

(135)
$$pagent\_select(\; Arg1 \;\&\; \left[\text{PROTO}\left[\begin{array}{ll}\text{VOLITION} & A1 \;\&\; 0..1\\ \text{SENTIENCE} & B1 \;\&\; 0..1\\ \text{CAUSE\_EVNT} & C1 \;\&\; 0..1\\ \text{MOVEMENT} & D1 \;\&\; 0..1\\ \text{REF\_EXISTS} & E1 \;\&\; 0..1\\ \text{PAGENT\_SUM} & (A1+B1+C1+D1+E1) \;\&\; X\end{array}\right]\right], \\ Arg2 \;\&\; \left[\text{PROTO}\left[\begin{array}{ll}\text{VOLITION} & A2 \;\&\; 0..1\\ \text{SENTIENCE} & B2 \;\&\; 0..1\\ \text{CAUSE\_EVNT} & C2 \;\&\; 0..1\\ \text{MOVEMENT} & D2 \;\&\; 0..1\\ \text{REF\_EXISTS} & E2 \;\&\; 0..1\\ \text{PAGENT\_SUM} & (A2+B2+C2+D2+E2) \;\&\; Y\end{array}\right]\right], \\ Arg2) : -Y \geq X.$$

Given the above entries for the traditional roles it is easy to see that if any argument is marked as an AGENT then this role will always be realised as the proto-agent. On the other hand, if any particular argument has the largest number of proto-agent

contributing properties then in that case either argument can be realised as the proto-agent.

We can use a similar mechanism for identifying the proto-patient role as given below. However, instead of introducing a new relational dependency *trans_p_patient* (resp. *ditrans_p_patient*) we combine both *trans_p_agent* (resp. *ditrans_p_agent*) and *trans_p_patient* (resp. *ditrans_p_patient*) into a single relational dependency *trans_p_select* (resp. *ditrans_p_select*) as shown below.

(136) $$intrans\_p\_select = \left[\text{SYN|ARGS } \exists : \left[\text{ROLE|PROTO } p\_agent\right]\right]$$

(137) $$intrans = intrans\_p\_select \ \& \ \ldots$$

(138) $$trans\_p\_select = \left[\text{SYN|ARGS}\left\{\boxed{1},\boxed{2}\right\}\right] : -$$
$$pagent\_select(\boxed{1},\boxed{2},\left[\text{ROLE|PROTO|NAME } p\_agent\right]),$$
$$ppatient\_select(\boxed{1},\boxed{2},\left[\text{ROLE|PROTO|NAME } p\_patient\right]).$$

(139) $$trans = trans\_p\_select \ \& \ \ldots$$

(140) $$ditrans\_p\_select = \left[\text{SYN|ARGS}\left\{\boxed{1},\boxed{2},\boxed{3}\right\}\right] : -$$
$$pagent\_select(\boxed{1},\boxed{2},\boxed{4}),$$
$$pagent\_select(\boxed{3},\boxed{4},\left[\text{ROLE|PROTO|NAME } p\_agent\right]),$$
$$ppatient\_select(\boxed{1},\boxed{2},\boxed{5}),$$
$$ppatient\_select(\boxed{3},\boxed{5},\left[\text{ROLE|PROTO|NAME } p\_patient\right]).$$

(141) $$ditrans = ditrans\_p\_select \ \& \ \ldots$$

(142)
$$ppatient\_select(\quad Arg1 \ \& \left[\text{PROTO}\begin{bmatrix}\text{CH\_STATE} & A1 \ \&\ 0..1\\ \text{INC\_THEME} & B1 \ \&\ 0..1\\ \text{CSLY\_AFTD} & C1 \ \&\ 0..1\\ \text{STATIONARY} & D1 \ \&\ 0..1\\ \text{REF\_NOT\_EXIST} & E1 \ \&\ 0..1\\ \text{PPATIENT\_SUM} & (A1+B1+C1+D1+E1) \ \&\ X\end{bmatrix}\right],$$
$$Arg2 \ \& \left[\text{PROTO}\begin{bmatrix}\text{CH\_STATE} & A2 \ \&\ 0..1\\ \text{INC\_THEME} & B2 \ \&\ 0..1\\ \text{CSLY\_AFTD} & C2 \ \&\ 0..1\\ \text{STATIONARY} & D2 \ \&\ 0..1\\ \text{REF\_NOT\_EXIST} & E2 \ \&\ 0..1\\ \text{PPATIENT\_SUM} & (A2+B2+C2+D2+E2) \ \&\ Y\end{bmatrix}\right],$$
$$Arg1) : -X \geq Y.$$

(143)

$$ppatient\_select\Big(\; Arg1 \;\&\; \left[\text{PROTO} \left[\begin{array}{ll} \text{CH\_STATE} & A1 \;\&\; 0..1 \\ \text{INC\_THEME} & B1 \;\&\; 0..1 \\ \text{CSLY\_AFTD} & C1 \;\&\; 0..1 \\ \text{STATIONARY} & D1 \;\&\; 0..1 \\ \text{REF\_NOT\_EXIST} & E1 \;\&\; 0..1 \\ \text{PPATIENT\_SUM} & (A1 + B1 + C1 + D1 + E1) \;\&\; X \end{array}\right]\right],$$

$$Arg2 \;\&\; \left[\text{PROTO} \left[\begin{array}{ll} \text{CH\_STATE} & A2 \;\&\; 0..1 \\ \text{INC\_THEME} & B2 \;\&\; 0..1 \\ \text{CSLY\_AFTD} & C2 \;\&\; 0..1 \\ \text{STATIONARY} & D2 \;\&\; 0..1 \\ \text{REF\_NOT\_EXIST} & E2 \;\&\; 0..1 \\ \text{PPATIENT\_SUM} & (A2 + B2 + C2 + D2 + E2) \;\&\; Y \end{array}\right]\right],$$

$$Arg1) : -Y \geq X.$$

With both proto-agent and proto-patient selection principles in force, arguments will have to compete for proto-role selection. If for instance, both arguments qualify for proto-agenthood but only one of them qualify for proto-patienthood then this will force the other argument to be realised as the proto-agent.

Our approach is similar in spirit to the approach taken in [Sanfillipo 90] which also provides a computational treatment of proto-role selection based on the proto-properties entailed by the meaning of verbal arguments. Sanfillipo's approach [Sanfillipo 90] is based on the idea of explicitly introducing a new sort symbol for every boolean combination of proto-agent (*resp.* proto-patient) properties. Thus in his approach PAT1, PAT2 and PAT3 would be a sorts equivalent to:

(144)

$$PAT1 = CH\_STATE \wedge CSLY\_AFTD \wedge STATIONARY$$

$$PAT2 = CH\_STATE \wedge CSLY\_AFTD \wedge \neg STATIONARY$$

$$PAT3 = \neg CH\_STATE \wedge CSLY\_AFTD \wedge STATIONARY$$

$$PAT4 = \neg CH\_STATE \wedge CSLY\_AFTD \wedge \neg STATIONARY$$

$$PAT5 = \neg CH\_STATE \wedge \neg CSLY\_AFTD \wedge STATIONARY$$

$$PAT6 = \neg CH\_STATE \wedge \neg CSLY\_AFTD \wedge \neg STATIONARY$$

Of course, the above definitions need not be stated within a typed feature formalism. What is stated instead is the following proto-patient hierarchy:

(145) $$\text{PAT1} > \left\{ \begin{array}{l} \text{PAT2} \\ \text{PAT3} \end{array} \right\} > \left\{ \begin{array}{l} \text{PAT4} \\ \text{PAT5} \end{array} \right\} > \text{PAT6}$$

The problem with this approach is that one needs 2^n sort symbols for n proto-properties. Furthermore, the linguist has to explicitly determine the partial ordering between the sort symbols. In constrast within our approach the linguist only need to provide a definition of thematic roles in terms of proto-properties.

Yet another difficulty with this approach is that there is no way to specify that a proto-property may or may not be entailed. Thus to specify that the traditional theme may or may not entail the MOVEMENT property, one may be forced to employ disjunction of sort symbols. Then thematic arguments that behave as themes will need to be specified as the disjunction $AGT1 \sqcup AGT2$ where $AGT1$ and $AGT2$ are defined as follows:

(146) $AGT1 = \neg VOLITION \wedge \neg SENTIENCE \wedge \neg CAUSE\_EVNT \wedge \neg MOVEMENT \wedge REF\_EXIST$

$AGT2 = \neg VOLITION \wedge \neg SENTIENCE \wedge \neg CAUSE\_EVNT \wedge MOVEMENT \wedge REF\_EXIST$

Once the sort hierarchy is in place Sanfillipo employs a disjunction of *negated* sort definitions to determine the proto-agent to be selected (see [Sanfillipo 90] *pp.* 145). The use of negation makes the specification very difficult to understand as it loses the benefit of a high-level specification.

6.2.4 Subject Selection

Once the argument proto-agent role has been identified it is straightforward to identify the *subject*. For non-passive verb forms the proto-agent will function as the subject. On the other hand for passive verb forms one of the *p_others* will function as the subject. We can state this constraint as follows:

(147)
$$select\_subject = \left[\text{SYN}\left[\begin{array}{ll}\text{HEAD|VFORM} & D(\neg pas, pas) \\ \text{ARGS} & \exists : \left[\text{ROLE}\left[\begin{array}{ll}\text{PROTO} & D(p\_agent, p\_others) \\ \text{SYN} & sub\end{array}\right]\right]\end{array}\right]\right]$$

(148) $main\_verbs = arg\_selection\_principles \ \& \ \ldots$

(149) $regular\_verbs = select\_subject \ \& \ \ldots$

We will require all main verbs to inherit the above *argument selection principle* and all regular main verbs to inherit the *subject selection principles.*

Direct Object Selection

Direct object selection will be similar to that of subject selection in that the proto-patient will be realised as the direct object in a non-passive construction. However in a passive construction any argument can be realised as the direct object.

(150)
$$direct\_object\_select = \left[\begin{array}{l}\text{SYN}\left[\begin{array}{ll}\text{HEAD|VFORM} & D(\neg pas, pas) \\ \text{ARGS} & \exists : \left[\text{ROLE}\left[\begin{array}{ll}\text{PROTO} & D(p\_patient, p\_role) \\ \text{SYN} & do\end{array}\right]\right]\end{array}\right]\end{array}\right]$$

(151) $trans = trans\_p\_select \;\&\; direct\_object\_select \;\&\; \ldots$

(152) $ditrans = ditrans\_p\_select \;\&\; direct\_object\_select \;\&\; \ldots$

Note that, as the above definitions indicate, direct object selection principles are not applicable to intransitive verbs and hence the direct object selection principle is only inherited by transitive and intransitive verbs.

6.3 A sign based treatment of DRT

In this section we provide a description of a reformulation of the logical treatment of DRT [Kamp 81] developed in [Johnson & Klein 86] so that it can be embedded within HPSG. This reformulation requires the use of the *union* operation over set descriptions which has not yet been investigated in this thesis. However we provide an outline of both an extended feature logic with a host of set operations including the union construct and the additional constraint solving ruled needed in the Appendix A.

We assume basic familiarity with the logical characterisation of DRT developed in [Johnson & Klein 86].

Within the framework adopted in [Johnson & Klein 86] the meaning of a linguistic expression α is a relation between a *preceding context* and a *following context i.e.*:

(153) Preceding-Context | α | Following-Context

Lexical items then express their meaning as a relation between a preceding context and a following context. Thus the meaning of *woman* can be expressed by the following definition where f stands for a discourse referent for a singular, female person [Johnson & Klein 86]:

(154) C | woman | $C \cup \{f\}$

Then, *anaphoric* dependencies can be treated by assuming that pronouns such as *her* are specified as:

(155) C | her | C iff $f \in C$

Thus the anaphoric dependency exhibited in sentences such (156) is accounted for.

(156) A woman$_i$ went home. She$_i$ was tired.

However, according to the theory of DRT lexical items are also allowed to introduce sub-contexts. Sub-contexts are introduced by *universal quantifiers* and govern the anaphoric dependencies exhibited in sentences such as (157).

(157) a. Every woman went home. She was sick.

b. Every woman who kissed a man$_i$ loved him$_i$.

The DRT representation of the sentences (157)a and (157)b are given in figure 6.18. To explain the contrast in the anaphoric behaviour of sentences (157)a and (157)b pronouns are allowed to access "higher" DRSs for finding compatible reference markers. A DRS A is higher than another DRS B if:

1. B is contained within A, or
2. B is a consequent of A, or
3. A is higher than some A' such that A' is higher than B

A DRS B is a consequent of A if A and B are related by the DRT implication A $\Rightarrow$ B.

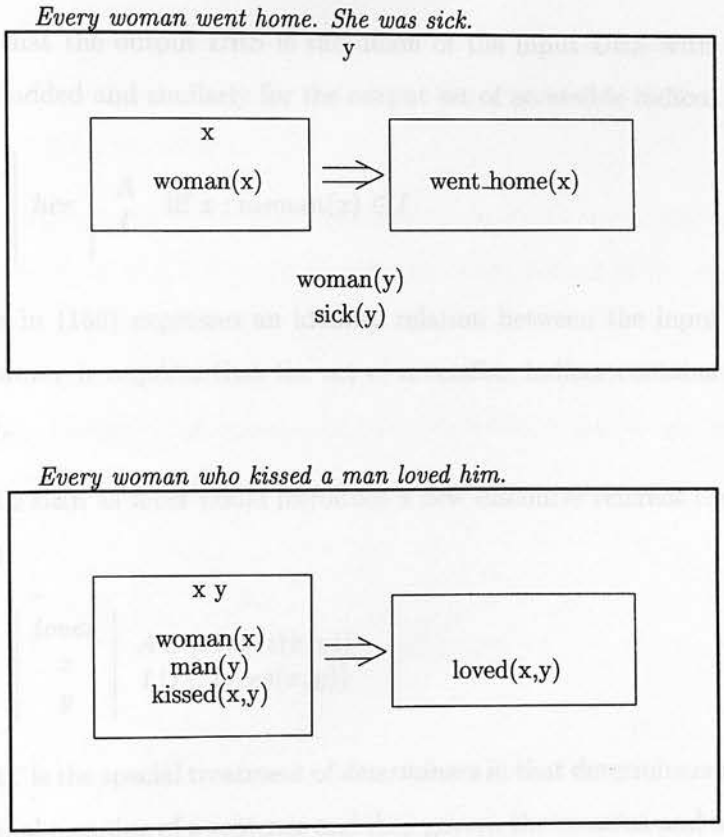


Figure 6.18: DRS representation for simple sentences

6.3.1 The specification of the meaning of lexical items

We represent a context by a pair (DRS, *accessible-referents*) where DRS is possibly a nested set of discourse referents and the set of accessible referents is a flat set of discourse referents. For any context (A, I) the set of accessible referents I are those accessible from the DRS A.

Lexical items then express their meaning as a relation between one or more contexts and sub-contexts. For instance, the meaning of *woman* and *her* can be specified as given in (158) and (159).

(158) $$\begin{array}{c|c|l} A & & A \cup \{x : woman(x)\} \\ I & woman & I \cup \{x : woman(x)\} \end{array}$$

The definition in (158) expresses a relation between an input context and an output

context such that the output DRS is the union of the input DRS with the referent $x : woman(x)$ added and similarly for the output set of accessible indices.

(159) $\begin{array}{|c|c|c|} A & & A \\ I & her & I \end{array}$ iff $x : woman(x) \in I$

The definition in (159) expresses an identity relation between the input and output contexts. However it requires that the set of accessible indices contains the referent $x : woman(x)$.

Similarly, verbs such as *loves* would introduce a new discourse referent : $loves(x, y)$ as given in (160).

(160) $\begin{array}{|c|c|l|} A & loves & A \cup \{: loves(x,y)\} \\ I & x & I \cup \{: loves(x,y)\} \\ & y & \end{array}$

Central to DRT is the special treatment of *determiners* in that determiners provide both shape of the final meaning of a sentence and they govern the creation and manipulation of sub-contexts.

It is fairly standard to think of determiner meanings as composed of a *restrictor* and *scope* [Barwise & Cooper 81] [Pereira & Shieber 87]. Both the *restrictor* and *scope* themselves represent linguistic meaning. This means that both the *restrictor* and *scope* are relations from a preceding context to a following context.

We then treat the meaning of a determiner as the relation between a preceding context, a following context and the preceding and following contexts of a restrictor and a scope.

The meanings of *a* and *every* are given in (161) and (162).

(161)
$$\begin{array}{|c|c|c|} & a & \\ A & \begin{array}{|c|c|c|} A & res & B \\ I & & M \end{array} & C \\ I & & O \\ & \begin{array}{|c|c|c|} B & scope & C \\ M & & O \end{array} & \end{array}$$

From the representation in (161) it should be clear that the determiner *a* "chains" the input context through the restrictor and then the scope thus characterising a notion of simple referent accumulation.

On the other hand, as shown in (162) the determiner *every* starts two new DRSs one each for the restrictor and the scope. It also chains the output of the set of accessible referents from the restrictor to the input of the scope. Finally, the output discourse referents consist of the input discourse referents together with two sub-DRSs joined by an implication. However, the contents of both the DRSs B and C are inaccessible for anaphora resolution to any higher DRSs.

(162)

$$\begin{array}{|c|c|c|c|c|}\hline & & \multicolumn{3}{c|}{every} & \\ \cline{3-5} A & \{\} & & B & \\ I & I & res & M & A \cup \{B \Rightarrow C\} \\ & & & & I \\ \cline{2-5} & \{\} & & C & \\ & M & scope & O & \\ \hline\end{array}$$

Given the above description of the semantic representation of lexical items, it is fairly straightforward to define HPSG sign based encodings.

6.3.2 HPSG encoding

We assume that the value of the SEMANTICS feature (or SEM feature for short) is specified at least for the structure described in (163).

(163)

$$\left[\text{SEM}\left[\begin{array}{ll}\text{DRS} & \left[\begin{array}{ll}\text{IN} & \top \\ \text{OUT} & \top\end{array}\right] \\ \text{INDS} & \left[\begin{array}{ll}\text{IN} & \top \\ \text{OUT} & \top\end{array}\right]\end{array}\right]\right]$$

The values of the IN and OUT features of the DRS feature will consist of DRSs. Each DRS is represented by a set consisting of discourse referents together with each embedded DRS being again represented by a set. The IN and OUT features of the INDS feature on the other hand will solely consist of a "flat" set of discourse referents.

Furthermore, determiners on the other hand will have the features RES and SCOPE specified as shown in the lexical entry for *a* given in (164).

(164)

$$a = \left[\begin{array}{l} \text{SYN|SUBCAT} \left\{ \left[\text{SEM } \boxed{1} \right] \right\} \\ \text{SEM} \left[\begin{array}{ll} \text{RES} & \left[\begin{array}{ll} \text{SEM } \boxed{1} & \left[\begin{array}{ll} \text{DRS} & \left[\begin{array}{ll} \text{IN} & A \\ \text{OUT} & B \end{array}\right] \\ \text{INDS} & \left[\begin{array}{ll} \text{IN} & I \\ \text{OUT} & M \end{array}\right] \end{array}\right] \end{array}\right] \\ \text{SCOPE} & \left[\begin{array}{ll} \text{SEM} & \left[\begin{array}{ll} \text{DRS} & \left[\begin{array}{ll} \text{IN} & B \\ \text{OUT} & C \end{array}\right] \\ \text{INDS} & \left[\begin{array}{ll} \text{IN} & M \\ \text{OUT} & O \end{array}\right] \end{array}\right] \end{array}\right] \\ \text{DRS} & \left[\begin{array}{ll} \text{IN} & A \\ \text{OUT} & C \end{array}\right] \\ \text{INDS} & \left[\begin{array}{ll} \text{IN} & I \\ \text{OUT} & O \end{array}\right] \end{array}\right] \end{array}\right]$$

The entry given in (164) is a straightforward encoding of the relationship (shown in (161)) between preceding context, following context and subcontexts that the determiner enforces.

However, we have made the assumption that determiners are heads and subcategorise for nouns contrary to the HPSG view in which nouns act as heads and subcategorise for determiners. We have also assumed that the restrictor of the determiner corresponds to the subcategorised noun (indicated by the index $\boxed{1}$ in (164)).

Similarly the sign based encoding of the semantic representation given in (162) of the determiner *every* is given in (165).

(165)

$$every = \begin{bmatrix} \text{SYN|SUBCAT} \left\{ \left[\text{SEM } \boxed{1} \right] \right\} \\ \text{SEM} \begin{bmatrix} \text{RES} & \text{SEM } \boxed{1} \begin{bmatrix} \text{DRS} \begin{bmatrix} \text{IN} & \{\} \\ \text{OUT} & B \end{bmatrix} \\ \text{INDS} \begin{bmatrix} \text{IN} & I \\ \text{OUT} & M \end{bmatrix} \end{bmatrix} \\ \text{SCOPE} \begin{bmatrix} \text{SEM} \begin{bmatrix} \text{DRS} \begin{bmatrix} \text{IN} & \{\} \\ \text{OUT} & C \end{bmatrix} \\ \text{INDS} \begin{bmatrix} \text{IN} & M \\ \text{OUT} & O \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{DRS} \begin{bmatrix} \text{IN} & A \\ \text{OUT} & A \cup \{B \Rightarrow C\} \end{bmatrix} \\ \text{INDS} \begin{bmatrix} \text{IN} & I \\ \text{OUT} & I \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The representation $B \Rightarrow C$ is a shorthand for some feature encoding such as $\begin{bmatrix} \text{ANTECEDENT } B \\ \text{CONSEQUENT } C \end{bmatrix}$.

The HPSG sign for *woman* is given in (166).

(166)

$$woman = \begin{bmatrix} \text{SEM} \begin{bmatrix} \text{DRS} \begin{bmatrix} \text{IN} & A \\ \text{OUT} & A \cup \boxed{1} \left\{ \left[\begin{bmatrix} \text{VAR} & x \\ \text{REST} \begin{bmatrix} \text{RELN} & woman \\ \text{INST} & x \end{bmatrix} \end{bmatrix} \right] \right\} \end{bmatrix} \\ \text{INDS} \begin{bmatrix} \text{IN} & I \\ \text{OUT} & I \cup \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

In order to treat lexical NPs such as proper names and personal pronouns in a semantically uniform fashion as quantified noun phrases we assume the modified specification of their meanings given in (167) and (168). This essentially is equivalent to type-raising as employed in UCG [Zeevat *et al* 87].

(167)

A					
I	$A \cup \{x : name(x, jane), x : woman(x)\}$	*jane*			B
	$I \cup \{x : name(x, jane), x : woman(x)\}$	*scope*	B		O
			O		

The idea in (167) now is that the referent for *jane* is added to the input context of the scope of the NP.

(168)

A					
I	A	*her*		B	such that $x : woman(x) \in I$
	I	*scope*	B	O	
			O		

However in (168) the input context is simply threaded through the scope of *her*.

The above relational descriptions can be encoded as HPSG signs as given in (169) and (170).

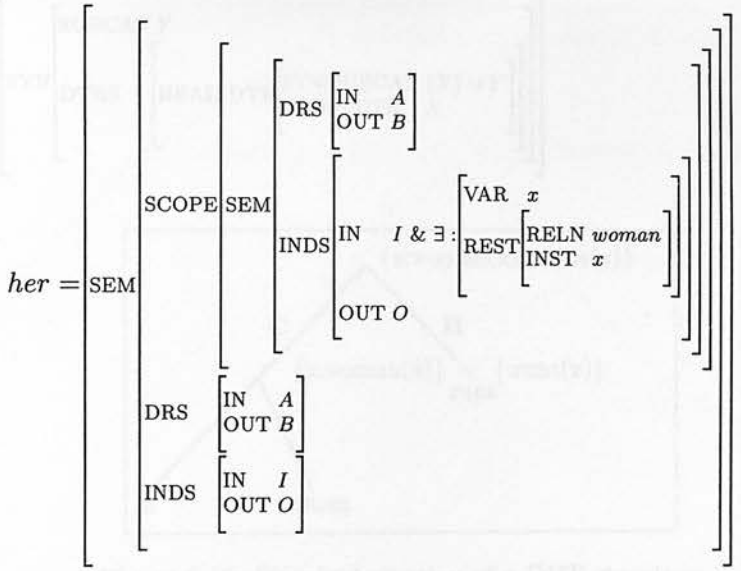
(169)

$$
jane = \left[\begin{array}{l}
\text{SEM} \left[\begin{array}{ll}
\text{SCOPE} & \text{SEM} \left[\begin{array}{ll}
\text{DRS} & \left[\begin{array}{ll} \text{IN} & A \cup \boxed{1} \\ \text{OUT} & B \end{array}\right] \\
\text{INDS} & \left[\begin{array}{ll} \text{IN} & I \cup \boxed{1} \\ \text{OUT} & O \end{array}\right]
\end{array}\right] \\
\text{DRS} & \left[\begin{array}{ll} \text{IN} & A \\ \text{OUT} & B \end{array}\right] \\
\text{INDS} & \left[\begin{array}{ll} \text{IN} & I \\ \text{OUT} & O \end{array}\right]
\end{array}\right]
\end{array}\right.
$$

$$
\left\{ \left[\begin{array}{ll} \text{VAR} & x \\ \text{REST} & \left[\begin{array}{ll} \text{RELN} & naming \\ \text{INST} & x \\ \text{NAME} & jane \end{array}\right] \end{array}\right], \left[\begin{array}{ll} \text{VAR} & x \\ \text{REST} & \left[\begin{array}{ll} \text{RELN} & woman \\ \text{INST} & x \end{array}\right] \end{array}\right] \right\}
$$

$$
\left.\begin{array}{l} \\ \\ \\ \\ \\ \\ \\ \end{array}\right]
$$

(170)



Finally we need some mechanism to properly instantiate the IN and OUT values of the contexts in a derivation. This is achieved by the *drs_principle* given in (171).

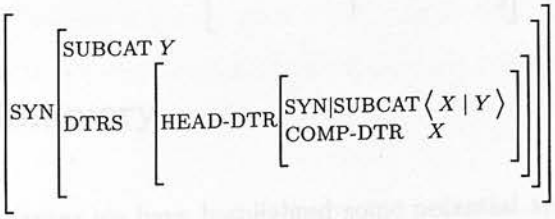
(171) $drs\_principle = \left[\text{SYN|DTRS}\left[\begin{array}{l}\text{HEAD-DTR|SEM} \quad \boxed{1} \\ \text{COMP-DTR|SEM|SCOPE} \quad \boxed{1}\end{array}\right]\right]$

$verbs \leq drs\_principle$

Essentially, the *drs_principle* instantiates the value of the "scope" of the complement daughter to the remainder of the derivation. Intuitively, this captures the effect that the remainder of the clause excluding the subcategorised noun is in the scope of the determiner. The inheritance specification $verbs \leq drs\_principle$ is intended to enforce the condition that the *drs_principle* is only obeyed by verbs.

We assume a binary branching version of the subcategorisation principle given in (172) or its set valued version given in (173).

(172)



(173)
$$\left[\begin{array}{l}\text{SYN}\left[\begin{array}{l}\text{SUBCAT } Y \\ \text{DTRS}\left[\text{HEAD-DTR}\left[\begin{array}{l}\text{SYN|SUBCAT } \{X\} \cup Y \\ \text{COMP-DTR } X\end{array}\right]\right]\end{array}\right]\end{array}\right]$$

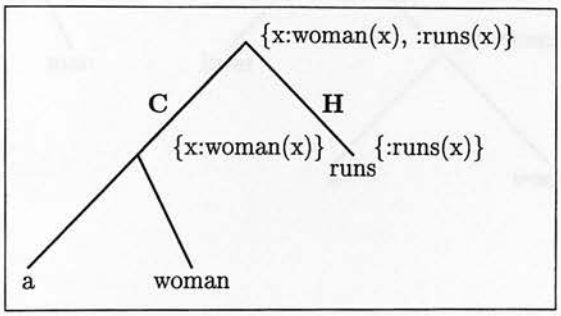


Figure 6.19: Simple derivation of a DRT structure

The *drs_principle* in conjunction with a binary branching subcategorisation principle (such as (173) or (172)) will be responsible for derivation structures of the kind depicted in figures 6.19 and 6.20. The variable bindings shown in figure 6.20 is the result of the *drs_principle*.

The definition of a *saturated sign* need to be modified to the one given in (174) which ensures that the value of the DRS|IN feature of a saturated sign is the empty set. For dealing with intersentential anaphora, the value of the INDS|OUT feature of the preceding sentence has to be unified with the value of the INDS|IN feature of the following sentence.

(174)
$$\textit{saturated\_sign} = \left[\begin{array}{l}\text{SYN|SUBCAT } \{\} \\ \text{SEM}\left[\begin{array}{l}\text{DRS}\left[\begin{array}{ll}\text{IN} & \{\} \\ \text{OUT} & R\end{array}\right] \\ \text{INDS}\left[\begin{array}{ll}\text{IN} & I \\ \text{OUT} & O\end{array}\right]\end{array}\right]\end{array}\right]$$

6.4 Summary

In this Chapter we have highlighted some potential applications for set descriptions. The examples described in this Chapter excludes the usage of CL-ONE for the speci-

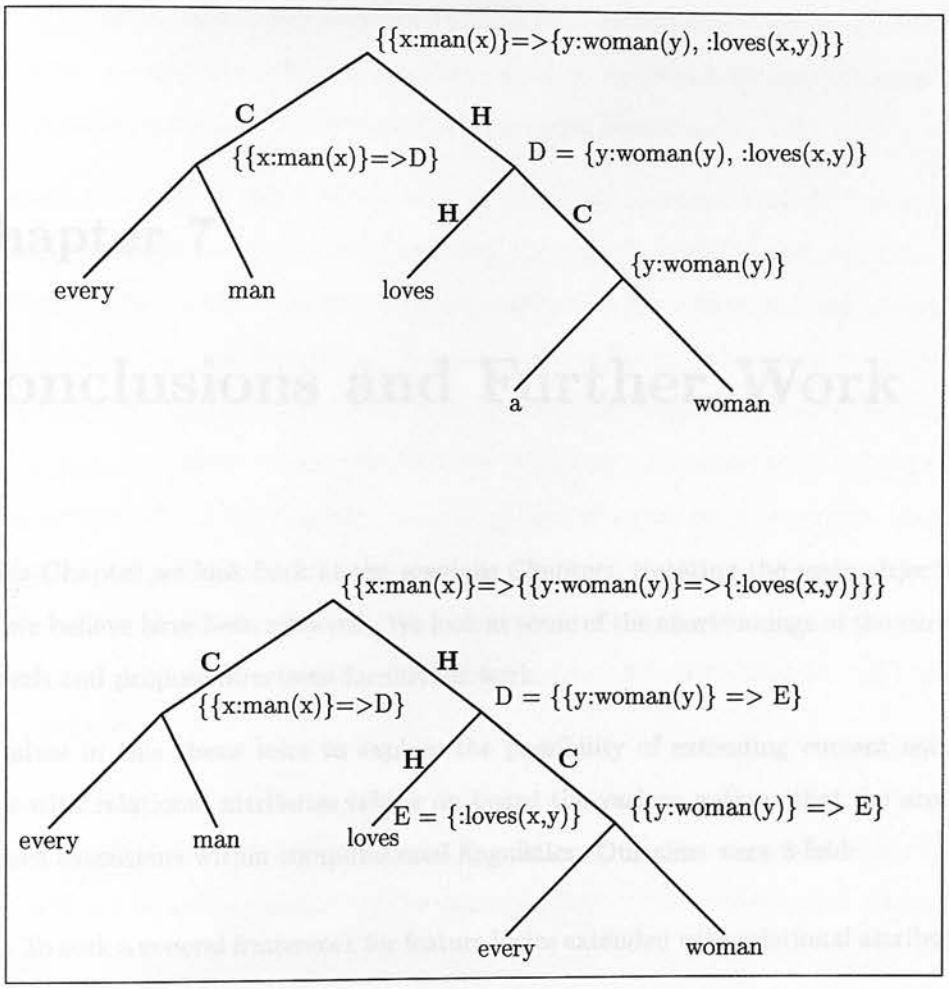


Figure 6.20: Further derivation of DRT structures

fication of word-order. These have been covered in Chapter 4.

Our examples show that set descriptions provide a high level of declarativity for stating linguistic knowledge and its usage lends towards more perspicious grammars.

However our treatment of DRT has also highlighted the fact the lack of the *union* construct over set descriptions is a serious drawback in CL-ONE. For this reason, we provide both a formulation and the necessary constraint solving rules for dealing with this additional construct in the next Chapter.

Chapter 7

Conclusions and Further Work

In this Chapter we look back at the previous Chapters, restating the main objectives that we believe have been achieved. We look at some of the shortcomings of the current research and propose directions for further work.

Our aims in this thesis were to explore the possibility of extending current feature logics with relational attributes taking on board the various notions that are around for such extensions within computational linguistics. Our aims were 3-fold:

1. To seek a general framework for feature logics extended with relational attributes.
2. To provide both a logical and computational basis for such extensions.
3. To demonstrate the applicability of the extended logic for constraint based grammars.

After presenting the basic concepts of *feature structures, feature logic, typed feature logics/formalisms, constraint based grammars* and *concept languages* in Chapter 1 we show in Chapter 2 that the addition of features and unquantified variables to the language $\mathcal{ALC}$ due to [Schmidt-Schauß & Smolka 91] does not render the logic undecidable. This shows that there is indeed a route for integrating feature logics with concept languages which form the backbone of A.I. based knowledge representation systems of the KL-ONE family. We believe that the integration of the KL-ONE family of languages with the typed feature logic based formalisms exemplified by formalisms such

as TFS [Zajac 92] and STUF [Dörre & Seiffert 91] is an important future direction for knowledge representation formalisms that intend to support both general knowledge representation tasks and the needs in computational linguistics.

Although there are already systems such as CLASSIC [Bordiga *et al* 89] that support features and path equations (over functional chains), the lack of support for variable co-references in CLASSIC makes it quite unattractive for a large number of applications in computational linguistics. The language $\mathcal{ALV}$ that we develop in Chapter 2 shows that it is indeed possible to integrate features and unquantified variables within a concept language without losing decidability. This result is somewhat surprising considering the fact that the straightforward integration of a variable-free concept language such as $\mathcal{ALC}$ with the relational analog of path equations known as *role value maps* causes undecidability [Schmidt-Schauß 89], hence blocking the obvious development of a decidable variable-free term language that supports relations, features and generic path equations.

In Chapter 3 we provided both a formalisation of the HPSG notion of set descriptions [Pollard & Moshier 90] which is different from the notion adopted in [Rounds 88] and a consistency checking method for the extended logic. Our formalisation shows that the logical machinery we develop in Chapter 2 is insufficient for encoding HPSG set descriptions. We demonstrate that set descriptions are no longer primitive if the so called *number restrictions* in concept languages such as BACK [von Luck *et al* 87] become available. This establishes a precise correspondence between set descriptions and concept languages enriched with number restrictions. However we also show that the translation of set descriptions into number restrictions introduces a large number of disjunctions and hence loses the benefit of a compact representation. For this reason, we build our constraint solving machinery by maintaining set descriptions. We also demonstrate how the various notions of extensionality that are commonly associated with sets can be accommodated within our semantics. This highlights potential issues that need to considered if consistency checking methods with extensional set descriptions are to be developed.

In Chapter 4 we explored the consequences of adding transitive relations to the posi-

tive fragment of the language $\mathcal{ALS}$ that we developed in the previous Chapter. We then showed that linear precedence constraints can be formalised as specialised constraints on transitive relations. We also highlighted the usefulness of our approach with examples from Dutch and German.

In Chapter 5 we showed a concrete design proposal for a linguistic formalism. Our formalism, dubbed CL-ONE, extends current typed feature logic based frameworks such as TFS [Zajac 92] and STUF [Dörre & Seiffert 91] with the provision of *set descriptions, linear precedence constraints, existential memberships, universal memberships, boolean constraints, limited distributed disjunctions, finite domain constraints* and *relational typing specifications.* We extended the constraint solving machinery for the language $\mathcal{ALO}$ developed in the previous Chapter to cope with the additional types of constraints introduced in CL-ONE. We believe that CL-ONE represents a new breed of formalisms that provides enhanced expressivity for broad spectrum knowledge representation tasks including those in computational linguistics.

In Chapters 6 we demonstrated some applications of CL-ONE for the specification of constraint based grammars. We showed how argument selection can be achieved with the aid of thematic information. Argument selection principles effectively translate thematic grids into subcategorisation frames. Novel to our approach is the idea that subject selection properties can be better determined by alluding to the proto-properties of thematic arguments following [Dowty 88]. We showed how proto-role selection can be encoded taking advantage of finite domain constraints available in CL-ONE.

Finally in Chapter 6 we provided a sign based reformulation of the DRT developed in [Johnson & Klein 86]. This shows that a HPSG based lexical treatment of DRT is feasible and indeed results in a clean and intuitively appealing representation of both semantic and discourse information within HPSG. We believe that this work can be extended for the treatment of VP-ellipsis [Gardent 93].

7.1 Some Shortcomings of the Current Approach

There are a fair number of shortcomings of the work reported in this thesis. We shall in this section try to highlight a number of these. This will provide the motivation for the direction of future work we outline in the next section.

On the formal side, although we have established both the logical formalisation and the computational basis for the logics that we explored in this thesis, we have not established any concrete complexity results for the logics we investigated. Thus for instance, although we know that consistency checking in $\mathcal{ALV}$ is PSPACE-hard we do not know whether is it PSPACE-easy too. Similarly we do not know whether the $\mathcal{ALS}$ sublanguage that excludes disjunction and negations has a NP-complete decision procedure (although we suspect this is to be the case from related results in set-valued term unification [A.Dovier *et al* 91]).

Secondly, although we have investigated abstract algorithms for consistency checking we have not investigated any concrete datastructures or algorithms. This is most needed for the language $\mathcal{ALO}$ which deals with transitive relations and linear precedence constraints in a way which is clearly inefficient.

Thirdly, our formulation of set descriptions is somewhat incomplete since we have not shown the consequences of adding the set union construct to the language $\mathcal{ALS}$ or at least the positive fragment of $\mathcal{ALS}$. The set union construct is desirable both for expressing a set based version of the subcategorisation principle and also in having the potential of simplifying the constraint solving machinery of the language $\mathcal{ALO}$ that deals with linear precedence constraints. In Appendix A we outline a formalisation and a complete set of constraint solving rules for dealing with set operations such as *union, intersection, subset* and *disjoint union.* This shows that a treatment of these set operations is indeed possible within a restricted language. However a more complete picture of the computational properties of these operations remains beyond the scope of this thesis.

Fourthly, we realise that the constraint solving machinery for dealing with set descrip-

tions introduces a fair degree of non-determinism. This will prevent the development of a really efficient constraint solving procedure. In section 7.2 we outline some methods for reducing the non-deterministic component that will be necessary for a efficient and practical implementation of a CL-ONE like language.

Finally, we have not been able to complete a realistic implementation of CL-ONE. This was mainly due to the fact that standard Prolog systems did not provide mechanisms such as constraint attachment and detachment from variables that is necessary for a good implementation. On the other hand, time limitations and the nature of the project prohibited more ambitious low-level implementations of CL-ONE. Nevertheless, most aspects of CL-ONE were verified in Prolog.

7.2 Other directions for Further Work

First and foremost, we believe that a realistic and efficient implementation of a CL-ONE like language can be achieved by reducing the non-deterministic component of the term language. There are two sources of non-determinism present in the positive fragment of $\mathcal{ALCS}$ which is included in CL-ONE. Firstly, it has to do with the interaction of set constraints of the form $x = f : \{x_1, \ldots, x_n\}$ with other set constraints of the form $x = f : \{y_1, \ldots, y_m\}$ which generates a large number of disjunctions (*see* Chapter 3). Secondly, it has to do with the interaction of set-membership constraints of the form $x \ni f\ y$ with set constraints which again creates a disjunction.

Although there appears to be no immediate solution to the first case there is a way around the second case. The idea is to turn *don't know non-determinism* into a *don't care non-determinism* by introducing *guarded constraints* [Saraswat & Rinard 90] [Hegner 91] [Aït-Kaci *et al* 92] [Smolka 91]. New constraints of the form $x \ni f\ G \mid y$ could be introduced where G is a *guarded constraint (set)*. The intended interpretation is that the variable y would be co-instantiated with one of the x_i's such that $x = f : \{x_1, \ldots, x_n\} \in C_s$ if x_i satisfies the guard G.

Thus for instance, given the system of constraints:

$$x = f : \{x_1, x_2\}, x_1 f y_1, y_1 : a, x_2 f y_2, y_2 : b, x \; \exists f \; \{\exists z_1 \; y f z_1, z_1 : a\} \mid y$$

The above constraints could be *determinately* reduced to:

$$x = f : \{x_1, x_2\}, x_1 f y_1, y_1 : a, x_2 f y_2, y_2 : a, x_1 = y, z_1 = y_1$$

This is because the variable y_1 satisfies the guard $\{\exists z_1 \; y f z_1, z_1 : a\}$ on the variable y.

However the use of guarded constraints although aiding efficient computation could be construed as a loss of declarativeness in the classical sense.

Note the existential quantification over z_1 which is necessary to code the existential nature of z_1. In general a guarded constraint set would be a constraint system of the form $\exists x_1 \ldots \exists x_n \; C_s$ where C_s is set of (ordinary) constraints.

The above proposals are a fairly conservative extension to the constraint language $\mathcal{S}_1$ which underlies the positive fragment of the term language $\mathcal{ALS}$. A general mechanism for handling guarded constraints in $\mathcal{S}_1$ could be formulated in the lines of [Smolka & Treinen 92].

Another possible line of further work is the investigation of the computational complexity of the positive sublanguage of $\mathcal{ALS}$ which should ultimately lead to a complexity analysis of the language $\mathcal{ALO}$. As related work in [Donini *et al* 91] shows, complexity analysis of sublanguages establishes the bounds of expressivity for tractable concept languages and is useful for guiding tractable extensions.

From a theoretical perspective it is desirable to establish the logical and computational properties of adding a set union construct to a complete fragment of $\mathcal{ALS}$ as opposed to just sublanguages. This should put to rest the issue of whether it is feasible to design realistic constraint solving algorithms with this extra construct when negative concept terms are included.

Known computational techniques for computing with transitive relations should be exploited for an efficient implementation of the constraint solving machinery for the language $\mathcal{ALO}$. This should take advantage of the availability of the set operations developed in Appendix A.

Finally the application of constraint languages such as CL-ONE for computational semantics has to be investigated. In Chapter 1 we showed with the aid of an example that a language such as $\mathcal{ALV}$ has potential application for the representation of situation theoretic structures. But we have not investigated the potential of the logics we explored in this thesis for the representation of situation theoretic structures and of semantic structures in general. Computational treatments of situation theoretic structures are just emerging [Nakashima *et al* 88] [Rupp 89] [Black 93]. Possible extensions to the constraint solving machinery explored in this thesis should be investigated to handle the needs within computational semantics.

Bibliography

[Aczel 88] Peter Aczel. *Non-well-founded Sets.* CSLI Lecture Notes. CSLI, 1988.

[A.Dovier *et al* 91] A.Dovier, E.G.Omodeo, E.Pontelli, and G.F.Rossi. {log}: A Logic Programming Language with Finite Sets. In K. Furukawa, editor, *Logic Programming: Proc. of the Eighth Int. Conf.* MIT Press, 1991.

[Aït-Kaci & Nasr 86] Hassan Aït-Kaci and Roger Nasr. LOGIN : A Logic Programming Language with Built-In Inheritance. *Journal of Logic Programming*, 3:185–215, 1986.

[Aït-Kaci & Podelski 91] Hassan Aït-Kaci and Andreas Podelski. An Overview of Life. In J. W. Schmidt and A. A. Stogny, editors, *Next Generation Information System Technology, Proceedings of the First International East/West Data Base Workshop, (Kiev, USSR–October 1990)*, pages 42–58. Springer-Verlag, 1991. LNCS 504.

[Aït-Kaci 84] Hassan Aït-Kaci. *A Lattice-Theoretic Approach to Computation Based on a Calculus of Partially-Ordered Type Structures.* Unpublished PhD thesis, University of Pennsylvania, 1984.

[Aït-Kaci *et al* 85] Hassan Aït-Kaci, Robert Boyer, Patrick Licoln, and Roger Nasr. Efficient Implementation of Lattice Operations. *TOPLAS*, 11(1):115–146, Jan 1985.

[Aït-Kaci *et al* 92] Hassan Aït-Kaci, G. Smolka, and R. Treinen. A Feature-based Constraint System for Logic Programming with Entailment. Research report, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, 6600 Saarbrücken 11, Germany, 1992.

[Baader 90] F. Baader. Terminological Cycles in KL-ONE based Knowledge Representation Languages. Research Report RR-90-01, German Research Center for Artifi-

cial Intelligence (DFKI), Stuhlsatzenhausweg 3, 6600 Saarbrücken 11, Germany, 1990.

[Baader 91] Franz Baader. Augmenting Concept Languages by Transitive Closure of Roles: An Alternative to Terminological Cycles. Research Report RR-90-13, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, 6600 Saarbrücken 11, Germany, December 1991.

[Baader *et al* 91] F. Baader, H.-J. Bürckert, B. Nebel, W. Nutt, and G. Smolka. On the Expressivity of Feature Logics with Negation, Functional Uncertainty and Sort Equations. Research Report RR-91-01, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, 6600 Saarbrücken 11, Germany, January 1991.

[Barwise & Cooper 81] Jon Barwise and Robin Cooper. Generalized quantifiers and natural language. *Linguistics and Philosophy*, 4(2):159–219, 1981.

[Barwise & Perry 83] J. Barwise and J. Perry. *Situations and Attitudes.* MIT Press, 1983.

[Barwise 88] John Barwise. A Situation in Logic-IV: On the Model Theory of Common Knowledge. CSLI Technical Report 122, CSLI, Stanford University, Stanford, CA, 1988.

[Bes & Gardent 89] Gabriel G Bes and Claire Gardent. French order without order. In *EACL89*, pages 249–255, April 1989.

[Black 93] A Black. *A Situation Theoretic approach to computational semantics.* Unpublished PhD thesis, University of Edinburgh, Edinburgh, UK., 1993.

[Boone 59] W. W. Boone. The word problem. *Annals of Mathematics*, 2(70):207–265, 1959.

[Bordiga *et al* 89] A. Bordiga, R.J. Brachman, D.L. McGuinness, and L.A. Resnick. CLASSIC: A structural data model for objects. In *1989 ACM SIGMOD International Conference on Management of Data*, pages 59–67, 1989.

[Brachman & Schmolze 85] R.J. Brachman and J.G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.

[Brachman *et al* 85] R. J. Brachman, Gilbert V. Pigman, and H. J. Levesque. An Essential Hybrid Reasoning System : Knowledge and Symbol Level Accounts in KRYPTON. In *9th International Joint Conference in Artificial Intelligence, (IJCAI), 1985, Los Angeles, CA*, pages 532–539, 1985.

[Carpenter 91] Bob Carpenter. Typed feature structures: A generalisation of first-order terms. In *International Logic Programming Symposium*, San Diego, 1991.

[Carpenter 93] Rob Carpenter. ALE:Attribute Logic Engine Users Guide, Version β. Technical report, Carnegie Mellon University, Pittsburgh, PA 15213, 1993.

[Carpenter *et al* 91] Bob Carpenter, Carl Pollard, and Alex Franz. The Specification and Implementation of Constraint-Based Unification Grammars. In *SIGPARSE International Workshop on Parsing Technologies*, Cancun, Mexico, 1991.

[Colmerauer 86] Alain Colmerauer. *Theoretical model of Prolog II*, chapter 1, pages 3–31. Ablex, Norwood, New Jersey, 1986.

[Dershowitz & Manna 78] Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. Report STAN-CS-78-651, Computer Science Department, Stanford University, Stanford, CA, March 1978.

[Dincbas *et al* 88] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, and T. Graf. Applications of CHIP to Industrial and Engineering Problems. In *First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, Tullahoma, Tennessee, June 1988.

[Donini *et al* 91] F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. Tractable Concept Languages. In *International Joint Conference in Artificial Intelligence, (IJCAI), 1991, Sydney, Australia*, pages 458–463, 1991.

[Dörre & Eisele 89] Jochen Dörre and Andreas Eisele. Determining consistency of feature terms with distributed disjunctions. In *GWAI-89, 13th German Workshop in Artificial Intelligence*, pages 270–279. Informatik Fachberichte 216, Springer, 1989.

[Dörre & Eisele 90] Jochen Dörre and Andreas Eisele. Feature logic with disjunctive unification. In *13th International Conference on Computational Linguistics, COLING-90, Helsinki*, volume 2, pages 100–105, 1990.

[Dörre & Rounds 90] Jochen Dörre and William C. Rounds. On subsumption and semiunification in feature algebras. In *5th Annual IEEE Symposium on Logic In Computer Science*, pages 300–310, Philadelphia, PA, June 1990. IEEE Computer Society Press.

[Dörre & Seiffert 91] Jochen Dörre and Roland Seiffert. Sorted feature terms and relational dependencies. IWBS Report 153, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, February 1991.

[Dörre *et al* 90] Jochen Dörre, Andreas Eisele, and Jürgen Wedekind. A Survey of Linguistically Motivated Extensions to Unification-Based Formalisms. Dyana, dynamic interpretation of natural language, espirit basic research action br3175, deliverable r3.1.a, Centre for Cognitive Science, University of Edinburgh, 2 Buccleuch Place, Edinburgh, EH8 9LW, U.K, Jan 1990.

[Dowty 88] David Dowty. Thematic Proto Roles, Subject Selection, and Lexical Semantic Defaults. In *1987 LSA Colloquium*, 1988. Preliminary DRAFT January 1988.

[Eisele & Dörre 90] Andreas Eisele and Jochen Dörre. A comprehensive unification formalism. Dyana internal memo, Institut für maschinelle Sprachverarbeitung, University of Stuttgart, September 1990.

[Emele & Zajac 90] Martin Emele and Rémi Zajac. A fix-point semantics of feature type systems. In *Second Workshop on Conditional and Typed Rewriting Systems - CTRS'90*, Montreal, June 1990.

[Engelkamp *et al* 92] J. Engelkamp, G. Erbach, and H. Uszkoreit. Handling linear precedence constraints by unification. In *Proceedings of the 30th Annual Meeting of the ACL*, pages 201–208, Newark, Delaware, 1992.

[Fillmore 68] Charles Fillmore. The Case for Case. In Emmon Bach and Robert T Harms, editors, *Universals in Linguistic Theory*, pages 1–90. Holt, Rinehart and Winston, New York, 1968. Papers presented at a symposium held at the University of Texas at Austin on April 13-15, 1967.

[Gardent 93] Claire Gardent. A unification-based approach to multiple VP Ellipsis resolution. In *EACL93*, pages 139–148, OTS, Utrecht, The Netherlands, April 1993.

[Garey & Johnson 79] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[Gazdar & Mellish 89] Gerald Gazdar and Chris Mellish. *Natural Language Processing in PROLOG - An Introduction to Computational Linguistics.* Addison-Wesley Publishing Company, 1989.

[Gazdar *et al* 85] G. Gazdar, E. Klein, G.K. Pullum, and I. Sag. *Generalised Phrase Structure Grammar.* Basil Blackwell, Oxford, U.K., 1985.

[Gurevich 66] Yuri Gurevich. The word problem for certain classes of semigroups. *Algebra and Logic*, 5:25–25, 1966.

[Hegner 91] S. Hegner. Horn extended feature structures: fast unification with negation and limited disjunction. In *Proceedings of the 5th conference of the European Chapter of the Association for Computational Linguistics*, pages 33–38, Berlin, Germany, 1991.

[Herzog & Rollinger 91] O. Herzog and C.-R. Rollinger, editors. *Text Understanding in LILOG.* Springer-Verlag, Berlin, Germany, 1991.

[Höhfeld & Smolka 88] Markus Höhfeld and Gert Smolka. Definite Relations over Constraint Languages. LILOG Report 53, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, October 1988. Accepted for the Journal of Logic Programming.

[Hollunder & Nutt 90] B. Hollunder and W. Nutt. Subsumption Algorithms for Concept Languages. Research Report RR-90-04, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, 6600 Saarbrücken 11, Germany, 1990.

[Holzbaur 90] Christian Holzbaur. *Specification of Constraint Based Inference Mechanisms through Extended Unification.* Unpublished PhD thesis, Dept. of Medical Cybernetics & Artificial Intelligence, University of Vienna, 1990.

[Johnson & Klein 86] Mark Johnson and Ewan Klein. Discourse, anaphora and parsing. In *11th International Conference on*

Computational Linguistics, COLING-86, pages 669–675, University of Bonn, Bonn, Germany, August 1986.

[Johnson 88] Mark Johnson. *Attribute-Value Logic and the Theory of Grammar.* CSLI Lecture Notes. CSLI, 1988.

[Kamp 81] Hans Kamp. A Theory of Truth and Semantic Representation. In *Formal Methods in the Study of Language*, volume 136, pages 277–322. Amsterdam: Mathematical Centre Tracts, 1981.

[Kaplan & Bresnan 82] R. M. Kaplan and J. Bresnan. *Lexical-Functional Grammar: A Formal System for grammatical representation*, pages 173–381. MIT Press, Cambridge, Mass., 1982.

[Kaplan & Maxwell III 89] R. M. Kaplan and J. T. Maxwell III. An algorithm for functional uncertainty. In *12th International Conference on Computational Linguistics, COLING-89*, pages 297–302, Budapest, Hungary, 1989.

[Karttunen 84] Lauri Karttunen. Features and values. In *10th International Conference on Computational Linguistics, COLING-84*, pages 28–33, Stanford University, Stanford, California, July 1984.

[Karttunen 86a] Lauri Karttunen. D-PATR: A Development Environment for Unification-Based Grammars. In *11th International Conference on Computational Linguistics, COLING-86*, pages 74–80, University of Bonn, Bonn, Germany, August 1986.

[Karttunen 86b] Lauri Karttunen. D-PATR: A Development Environment for Unification-Based Grammars. Csli report, Center for the Study of Language and Information, Stanford University, Stanford, California, 1986.

[Kasper & Rounds 86] Robert Kasper and William Rounds. A logical semantics for feature structures. In *24th Annual Meeting of the Association for Computational Linguistics, Columbia University, New York*, pages 257–265, 1986.

[Kasper 87] Robert Kasper. A unification method for disjunctive feature descriptions. In *25th Annual Meeting of the Association for Computational Linguistics*, pages 235–242, Stanford, CA, 1987.

[Kobsa 89] Alfred Kobsa. The SB-ONE knowledge representation workbench. In *Preprints of the Workshop on Formal*

Aspects of Semantic Networks. Two Harbors, CA., Febraury 1989.

[Lloyd 84] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1984.

[MacGregor & Bates 87] R. MacGregor and R. Bates. The LOOM Knowledge Representation Language. Technical Report ISI/RS-87-188, University of Southern California, Information Science Institute, Marina del Rey, CA, 1987.

[MacGregor 88] R. MacGregor. A deductive pattern matcher. In *Seventh National Conference of the American Association for Artificial Intelligence*, pages 403–408, AAAI, Menlo Park, CA, 1988.

[Main 87] Michael G. Main. A powerdomain primer. *EATCS Bulletin, European Association for Theoretical Computer*, (33):115–147, October 1987.

[Mann & Mathiessen 85] William C. Mann and Christian I. M. I. Mathiessen. Demonstration of the Nigel text generation computer program. In James D. Benson and William S. Greaves, editors, *Systemic Perspectives on Discourse, Volume 1.* Ablex, Norwood, NJ, 1985.

[Moser 83] M. G. Moser. An overview of NIKL, the new implementation of KL-ONE. In *Research in Knowledge Representation and Natural Language Understanding, BBN Report No. 5421*, pages 7–26. Bolt, Beranek and Newman Inc., Cambridge, Mass., 1983.

[Moshier & Rounds 87] M. D. Moshier and W. C. Rounds. A logic for partially specified data structures. In *ACM Symposium on the Principles of Programming Languages*, Munich, Germany, 1987. Association for Computing Machinery.

[Nakashima *et al* 88] H. Nakashima, H. Suziki, P-K. Halvorsen, and S. Peters. Towards a Computational Interpretation of Situation Theory. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 489–498. ICOT, 1988.

[Nebel & Smolka 89] Bernhard Nebel and Gert Smolka. *Representation and Reasoning with Attributive Descriptions*, pages 112–139. Springer-Verlag, Berlin, Germany, November 1989.

[Nebel & Smolka 91] Bernard Nebel and Gert Smolka. Attributive Description Formalisms and the Rest of the World. Research

Report RR-91-15, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, 6600 Saarbrücken 11, Germany, May 1991.

[Nebel & Sondheimer 86] Bernhard Nebel and Norman K. Sondheimer. NIGEL gets to know logic: an experiment in natural language generation taking a logical knowledge-based view. In C. Rollinger and W. Horn, editors, *Proceedings of the GWAI-86*. Springer-Verlag, Berlin, Germany, 1986.

[Nebel 90] Bernhard Nebel. *Reasoning and Revision in Hybrid Representation Systems*. Lecture Notes in Artificial Intelligence, No. 422. Springer-Verlag, 1990.

[Nebel 91] Bernhard Nebel. *Terminological Cycles: Semantics and Computational Properties*, pages 331–361. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1991.

[Novikov 55] P. S. Novikov. On the algorithmic undecidability of the word problem in group theory. Technical report, Trudy Mat. Inst., Steklov. 14, Izdat. Nauk SSSR, Moscow, 1955.

[Owsnicki-Klewe 88] Bernd Owsnicki-Klewe. Configuration as a Consistency Maintenance Task. In *GWAI-88*, pages 77–87. Springer, Berlin, September 1988.

[Partee *et al* 90] Barbara H. Partee, Alice ter Meulen, and Robert E. Wall. *Mathematical Methods in Linguistics*. Studies in Linguistics and Philosophy. Kluwer Academic Publishers, 1990.

[Patel-Schneider 84] P.F. Patel-Schneider. Small can be beautiful in knowledge representation. In *Proceedings of the IEEE Workshop on Principles of Knowledge-representation*, pages 11–16, Denver, Colorado, 1984.

[Peltason 87] Christof Peltason. The Scheme of Posidonius - Using Taxonomic Reasoning in Design. In *Proc. Second Int. Conf. on Applications of Artificial Intelligence in Engineering*, pages 299–314, Cambridge, Mass., August 1987.

[Peltason *et al* 91] C. Peltason, K. von Luck, and C. Kindermann (Org.). Terminological Logic Users Workshop. Kit report, Department of Computer Science, Technische Universität Berlin, Berlin, Germany, October 1991.

[Pereira & Shieber 87] Fernando C.N. Pereira and Stuart Shieber. *Prolog and Natural Language Analysis.* CSLI Lecture Notes. CSLI, 1987.

[Pletat & von Luck 89] Udo Pletat and Kai von Luck. *Knowledge Representation in LILOG*, pages 140–164. Springer-Verlag, Berlin, Germany, November 1989.

[Pollard & Moshier 90] Carl J. Pollard and M. Drew Moshier. Unifying Partial Descriptions of Sets. In P. Hanson, editor, *Information, Language and Cognition*, volume 1 of *Vancouver Studies in Cognitive Science.* University of British Columbia Press, Vancouver, 1990.

[Pollard & Sag 87] Carl Pollard and Ivan Sag. *Information based syntax and semantics, VOL I.* CSLI Lecture Notes. CSLI, 1987.

[Pollard & Sag 92] Carl Pollard and Ivan Sag. *Head-driven Phrase Structure Grammar, VOL II.* MIT Press, 1992. Forthcoming.

[Radford 88] Andrew Radford. *Transformational Grammar, A First Course.* Cambridge Textbooks in Linguistics. Cambridge University Press, 1988.

[Reape 89] Mike Reape. A logical treatment of semi-free word order and bounded discontinuous constituency. In *EACL89*, pages 103–110, 1989.

[Reape 93] Mike Reape. *Getting Things in Order.* 1993. Forthcoming.

[Reiter & Mellish 92] Ehud Reiter and Chris Mellish. Using classification to generate text. In *Proceedings of the ACL-92*, 1992.

[Rounds 88] William C. Rounds. Set Values for Unification-Based Grammar Formalisms and Logic Programming. Technical report, CSLI: Stanford University, 1988.

[Rupp 89] C. Rupp. Situation semantics and machine translation. In *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, pages 308–318, Manchester, UK., 1989.

[Sanfillipo 90] Antonio Sanfillipo. *Grammatical Relations, Thematic Roles and Verb Semantics.* Unpublished PhD thesis, University of Edinburgh, 1990.

[Saraswat & Rinard 90] V. Saraswat and M. Rinard. Concurrent Constraint Programming. In *Proceedings of the 7th ACM Symposium on the Principles of Programming Languages*, pages 232–245, San Francisco, CA, January 1990. Association for Computing Machinery.

[Schild 91] Klaus Schild. A correspondence theory for teminological logics: Preliminary report. In *12th International Joint Conference on Artificial Intelligence*, Sydney, Australia, 1991.

[Schmidt-Schauß & Smolka 91] Manfred Schmidt-Schauß and Gert Smolka. Attributive Concept Descriptions with Unions and Complements. *Artificial Intelligence*, 48:1–26, 1991. Also available as IWBS Report 68, IBM Germany, Scientific Center, IWBS, Stuttgart, Germary, June 1989.

[Schmidt-Schauß 89] Manfred Schmidt-Schauß' Subsumption in KL-ONE is undecidable. In *First International Conference on Principles of Knowledge Representation and Reasoning, KR' 89, Toronto, Canada*, pages 421–431, May 1989.

[Schmolze 89] J.G. Schmolze. The Language and Semantics of NIKL. Technical Report 89-4, Department of Computer Science, Tufts University, Medford, MA, 1989.

[Shensa 89] M. J. Shensa. A computational structure for the propositional calculus. In *11th International Joint Conference on Artificial Intelligence*, pages 384–388, Detroit, Michigan, 1989.

[Shieber 84] S. Shieber. The design of a computer language for linguistic information. In *Proceedings of the 10th International Conference on Computational Linguistics/22nd Annual Conference of the Association for Computational Linguistics*, pages 362–366, Stanford University, California, 1984.

[Shieber 86] Stuart Shieber. *An Introduction to Unification Based Approaches to Grammar*. CSLI Lecture Notes. CSLI, 1986.

[Smolka & Treinen 92] G. Smolka and R. Treinen. Records for Logic Programming. Research Report RR-92-23, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, 6600 Saarbrücken 11, Germany, August 1992.

[Smolka 88] Gert Smolka. A Feature Logic with Subsorts. LILOG Report 33, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, May 1988. To appear in: J. Wedekind and C. Rohrer (eds.), Unification in Grammar; The MIT Press, 1991.

[Smolka 89] Gert Smolka. Feature Constraint Logics for Unification Grammars. IWBS Report 93, IWBS, IBM Deutschland, Postfach 80 08 80, 7000 Stuttgart 80, Germany, November 1989. To appear in the Journal of Logic Programming in 1991.

[Smolka 91] Gert Smolka. Residuation and Guarded Rules for Constraint Logic Programming. Research Report RR-91-13, German Research Center for Artificial Intelligence (DFKI), Stuhlsatzenhausweg 3, 6600 Saarbrücken 11, Germany, May 1991. Also available as PRL Research Report 12, Digital, 85 avenue Victor Hugo, 92563 Rueil-Malmaison Cedex, France.

[Steedman 85] Mark Steedman. Dependency and coordination in a grammar of dutch and english. Research Paper 248, Department of Artificial Intelligence, Edinburgh, 80 South Bridge, Edinburgh EH1 1HN, U.K., 1985.

[Uszkoreit 85] Hans Uszkoreit. Constraints on order. Technical Note 364, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025, October 1985.

[Van Hentenryck & Dincbas 86] P. Van Hentenryck and M. Dincbas. Domains in Logic Programming. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-86), Philadelphia*, pages 759–765, August 1986.

[Van Hentenryck & Dincbas 87] P. Van Hentenryck and M. Dincbas. Forward Checking in Logic Programming. In J-L. Lassez, editor, *Logic Programming: Proceedings of the 4th International Conference (Melbourne)*, pages 229–256. MIT Press, May 1987.

[Van Hentenryck 89] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, 1989.

[Vilain 85] Mark B. Vilain. The restricted language architecture of a hybrid representation system. In *9th International Joint Conference in Artificial Intelligence, (IJCAI), 1985, Los Angeles, CA*, pages 547–551, 1985.

[von Luck *et al* 87] K. von Luck, B. Nebel, C. Peltason, and A. Schmiedel. The Anatomy of the BACK System. KIT Report 41,

Department of Computer Science, Technische Universität Berlin, Berlin, Germany, 1987.

[Williams 81] Edwin Williams. Argument Structure and Morphology. *The Linguistic Review*, 81-114, 1981.

[Zajac 90a] Rémi Zajac. A relational approach to translation. In *Proceedings, 3rd International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Language*, Austin, Texas, 1990.

[Zajac 90b] Rémi Zajac. Semantics of typed feature structures. In *International Workshop on Constraint Based Formalisms for Natural Language Generation*, Bad Teinach, Germany, 1990.

[Zajac 92] Rémi Zajac. Inheritance and Constraint-Based Grammar Formalisms. *Computational Linguistics*, 18(2):159–182, 1992.

[Zeevat *et al* 87] Henk Zeevat, Ewan Klein, and Jonathan Calder. An introduction to Unification Categorial Grammar. In Nicholas J. Haddock, Ewan Klein, Glyn Morrill, editor, *Edinburgh Working Papers in Cognitive Science*, volume 1, pages 195–222. Centre for Cognitive Science, University of Edinburgh, 1987.

Appendix A

Augmenting set descriptions with set operations

Our aim in this section is to provide and formalisation and constraint solving machinery for additional operations on set descriptions namely - *subset, union* and *intersection* operations. These additions we hope will complete the support for set descriptions.

Our idea is to restrict these set operations to variables only and hence restrict the possible usage of these contructs. The syntax for the additional term constructs is given below:

$$
\begin{array}{lll}
S,T \longrightarrow \ldots\ldots & | \, f : g(x) \cup h(y) & \text{union} \\
& | \, f : g(x) \cap h(y) & \text{intersection} \\
& | \, f :\supseteq g(x) & \text{subset} \\
& | \, f : g(x) \uplus h(y) & \text{disjoint union}
\end{array}
$$

An interpretation for the above new constructs can be provided by the following definitions:

$$[\![f : g(x) \cup h(y)]\!]^{\mathcal{I},\alpha} = \{e \in \mathcal{U}^I \mid f^I(e) = g^I(\alpha(x)) \cup h^I(\alpha(y))\}$$

$$[\![f : g(x) \cap h(y)]\!]^{\mathcal{I},\alpha} = \{e \in \mathcal{U}^I \mid f^I(e) = g^I(\alpha(x)) \cap h^I(\alpha(y))\}$$

$$[\![f :\supseteq g(x)]\!]^{\mathcal{I},\alpha} = \{e \in \mathcal{U}^I \mid f^I(e) \supseteq g^I(\alpha(x))\}$$

$[\![f : g(x) \uplus h(y)]\!]^{\mathcal{I},\alpha} =$

- $\emptyset$ if $g^I(\alpha(x)) \cap h^I(\alpha(y)) \neq \emptyset$
- $\{e \in \mathcal{U}^I \mid f^I(e) = g^I(\alpha(x)) \cup h^I(\alpha(y))\}$ if $g^I(\alpha(x)) \cap h^I(\alpha(y)) = \emptyset$

Thus the term $f : g(x) \cup h(y)$ denotes the collection of all objects whose f-value is the union of the g-values of (the object denoted by) x and h-values of (the object denoted by) y. The denotations of $f : g(x) \cap h(y)$ and $f :\supseteq g(x)$ are analogous.

The denotation of the term $f : g(x) \uplus h(y)$ is identical to that of $f : g(x) \cup h(y)$ except when the g-values of x and h-values of y are not disjoint. In this case the denotation of $f : g(x) \uplus h(y)$ is the empty set.

A.1 Constraint solving with Set operations

The idea behind constraint solving with set operations is to add new relations of the form $x \; \exists f \; y$ between existing variables whenever this is licenced by the semantics of existing constraints. The set of constraints will be in normal form when no further relations need to be added.

The constraint solving rules are presented in figure A.21.

Recollect from Chapter 3 that the **f-successors** of x in a given constraint system C_s denoted by $succ(x, f)$ is given by the definition:

$$succ(x, f) = \{y \mid xfy \in C_s \text{ or } x \; \exists f \; y \in C_s \text{ or } x = f : \{\ldots, y, \ldots\} \in C_s\}$$

We assume that containment constraints of the form $x \dot{\sqsubseteq} T$ (see Chapter 2) are decomposed by the following decomposition rules prior to the application of the constraint solving rules:

$$x \dot{\sqsubseteq} T \longrightarrow x = T \text{ if } T \text{ is a set operation}$$

Rule ($\subseteq$) adds $x \; \exists f \; y_i$ whenever we know that $xGy_i, G \in \{g, \exists g\}$ and every g-value of y is also a f-value of x.

Rules ($\cup Left$) and ($\cup Right$) simply assert that $x = f : g(y) \; \theta \; h(z)$ (where $\theta \in \{\cup, \uplus\}$) implies both $x = f :\supseteq g(y)$ and $x = f :\supseteq h(z)$.

Rule ($\cup Down$) is non-deterministic and creates two choice points. This happens when $x = f : g(y) \; \theta \; h(z), xFx_i, F \in \{f, \exists f\}, \theta \in \{\cup, \uplus\}$ is known such that both $yGx_i \notin C_s$ and $zHx_i \notin C_s$ where $G \in \{g, \exists g\}$ and $H \in \{h, \exists h\}$. In this case, it follows that either $y \; \exists g \; x_i$ or $z \; \exists h \; x_i$ need to be added non-deterministically.

Rule ($\cap Down$) ensures that every f-value of x which is also the g-value of y and h-value of z just in case $x = f : g(y) \cap h(z)$ is known.

Rule ($\cap Up$) ensures that every variable which is in the intersection of the g-values of y and h-values of z is also an f-value of x just in case $x = f : g(y) \cap h(z)$ is known.

The definition of *clash* (see Chapter 3) need to be extended with the following additional condition for dealing with the *disjoint union* construct.

Definition A.1.1 [Additional clash condition] *A constraint system C_s contains a clash if both:*

- $x = f : g(y) \uplus h(z) \in C_s$ *and*
- *there exists w such that $w \in succ(y, g)$ and $w \in succ(z, h)$*

Termination of the above set of constraint solving rules can be easily demonstrated if we add the above set operation constructs to the sublanguage of $\mathcal{ALO}$ (see Chapter 4) which restricts terms of the form $\forall f : T$ to $\forall f : \top$. In other words, we allow the following syntax:

$$
\begin{array}{rcl}
S, T & \longrightarrow & x \mid a \mid c \mid C \mid \exists f : T \mid \forall f : \top \mid S \sqcap T \mid S \sqcup T \mid f : \{T_1, \ldots, T_n\} \mid \\
& & p^*(x) \times q^*(y) \dot{\sqsubseteq} r \mid p^*(x) \times q^*(y) \dot{\sqsubseteq} r^+ \\
& & \mid f : g(x) \cup h(y) \\
& & \mid f : g(x) \cap h(y) \\
& & \mid f :\supseteq g(x) \\
& & \mid f : g(x) \uplus h(y)
\end{array}
$$

where $r, f \in \mathcal{F}$, $r \in \mathcal{F}i$ and $f \notin \mathcal{F}i$.

For the above sublanguage the constraint solving rules do not introduce more than a finite number of new variables generated intially by the Phase 2 decomposition rules (see Chapter 2). Termination follows since no further new variables are generated and the number of variables a given relation can relate is bounded by the number of variables in the constraint system.

However termination properties for the language without the above restriction has not been investigated.

Extended Constraint Solving

$(\subseteq)$ $\{x = f :\supseteq g(y)\} \cup C_s \longrightarrow \{x = f :\supseteq g(y), x\ \exists f\ y_i\} \cup C_s$
if:

- $y_i \in succ(x, g)$
- $y_i \notin succ(x, f)$

$(\cup Left)$ $\{x = f : g(y)\ \theta\ h(z)\} \cup C_s \longrightarrow \{x = f : g(y)\ \theta\ h(z), x = f :\supseteq g(y)\} \cup C_s$
if $x = f :\supseteq g(y) \notin C_s$ and $\theta \in \{\cup, \uplus\}$

$(\cup Right)$ $\{x = f : g(y)\ \theta\ h(z)\} \cup C_s \longrightarrow \{x = f : g(y)\ \theta\ h(z), x = f :\supseteq h(z)\} \cup C_s$
if $x = f :\supseteq h(z) \notin C_s$ and $\theta \in \{\cup, \uplus\}$

$(\cup Down)$ $\{x = f : g(y)\ \theta\ h(z)\} \cup C_s$
$\longrightarrow$
$\{x = f : g(y)\ \theta\ h(z), y\ \exists g\ x_i \mid z\ \exists h\ x_i\} \cup C_s$
if:

- $\theta \in \{\cup, \uplus\}$
- $x_i \in succ(x, f)$
- $x_i \notin succ(y, g)$ and
- $x_i \notin succ(z, h)$

$(\cap Down)$ $\{x = f : g(y) \cap h(z)\} \cup C_s$
$\longrightarrow$
$\{x = f : g(y) \cap h(z), y\ \exists g\ x_i, z\ \exists h\ x_i\} \cup C_s$
if:

- $x_i \in succ(x, f)$ and
- $x_i \notin succ(y, g)$ or $x_i \notin succ(z, h)$ and

$(\cap Up)$ $\{x = f : g(y) \cap h(z), yGx_i, zHx_i\} \cup C_s$
$\longrightarrow$
$\{x = f : g(y) \cap h(z), x\ \exists f\ x_i, yGx_i, zHx_i\} \cup C_s$
if:

- $x_i \in succ(y, g)$
- $x_i \in succ(z, h)$ and
- $x_i \notin succ(x, f)$

Figure A.21: Constraint solving with set operations

Appendix B

Termination Proof

In this appendix we show that our normalisation rules for the language $\mathcal{L}_2$ described in Chapter 2 terminate. This means that consistency of $\mathcal{ALV}$ terms is decidable.

As remarked in Chapter 2, intuitively speaking, it is clear that both Stage 1 and Stage 2 rules terminate in linear time. For Stage 3 rules, if we omit rule **BExists** then there would be no rules that generate new variables. Since there are a fixed number of big variables and a fixed number of constraints of the form $X \dot{\sqsubseteq} \Gamma$ it is intuitively clear that this process must also terminate.

The difficulty arises when we take into account rule **BExists** since it adds new variables.

Our proof utilises a complexity measure based on multiset orderings due to [Dershowitz & Manna 78] that is sensitive to the degree to which a given constraint system is in normal form.

Definition B.0.2 [Lexicographic Ordering] *Given well-founded sets* $(X_1, \succ_1), \ldots, (X_n, \succ_n)$ *a lexicographic ordering between tuples* $(A_1, \ldots, A_n)$ *and* $(B_1, \ldots, B_n)$ *where* $1 \leq i \leq n : A_i \in X_i$ *and* $1 \leq i \leq n : B_i \in X_i$ *is defined as follows:*

$(A_1, \ldots, A_n) \succ\!\succ (B_1, \ldots, B_n)$ *if*

there exists $i : 1 \leq i < n$ *such that*

$A_1 = B_1, \ldots, A_i = B_i$ *and*

$A_{i+1} \succ_{i+1} B_{i+1}$

We shall be implicitly assuming that only finite constraint systems are considered.

Theorem B.0.3 [Termination] *There is no infinite chain of rule applications issuing from any completion of the normalisation rules on any finite $\mathcal{L}_2$ constraint system.*

Proof: We shall establish the termination claim via a series of definitions and lemmas that progressively defines and verifies our complexity measure.

Let K be a function that maps any constraint system C_s to a tuple $K(C_s) = (\Delta_T, \Delta_{\dot{\sqsubseteq}}, \Delta_c)$ where:

- Δ_T is a multiset consisting of the *size* of each constraint of the form $\overline{X} \dot{\sqsubseteq} \Gamma \in C_s$ *excluding* constraints of the form $x \dot{\sqsubseteq} y$ and $x \dot{\sqsubseteq} X$. The *size* of each term $\overline{X} \dot{\sqsubseteq} \Gamma$ as described above is then calculated as follows:

  $size(\overline{X} \dot{\sqsubseteq} \Gamma) = 1 + size(\Gamma)$

  $size(\exists f : \Gamma) = size(\forall f : \Gamma) = 1 + size(\Gamma)$

  $size(X) = 3$

  $size(c) = 2$ (Note that this includes atoms)

  $size(C) = 1$

  $size(x) = 1$

  $size(S \sqcap T) = size(S) + size(T)$

  $size(S \sqcup T) = size(S) + size(T)$

  $size(\neg T) = (size(T) \times size(T)) + 1$

- $\Delta_{\dot{\sqsubseteq}}$ is a multiset of ordered pairs. The precise definition of $\Delta_{\dot{\sqsubseteq}}$ will be established as we progress through the proof.

- Δ_c is a multiset consisting of the complexities of all the $\mathcal{L}_1$ constraints as given in 2.4.3 plus the complexity of constraints of the form $x \dot{\sqsubseteq} y$ given by:

$$|x \dot{\sqsubseteq} y| = 8$$

The well-founded ordering between any two tuples $(\Delta_T, \Delta_{\dot{\sqsubseteq}}, \Delta_c)$ and $(\Delta'_T, \Delta'_{\dot{\sqsubseteq}}, \Delta'_c)$ is given by the lexicographic ordering.

The manner in which the above complexity measure is reduced by our consistency checking rules can be summarised as follows:

1. All Stage 1 and Stage 2 rules decrease Δ_T

2. For Stage 3 rules the measures affected are as follows:

   (a) All Group 0 rules decrease Δ_T

   (b) All Group 1 rules decrease Δ_c only

   (c) All Group 2 rules decrease $\Delta_{\dot{\sqsubseteq}}$

Lemma B.0.4 *The measure Δ_T is decreased by the application of:*

1. *any Stage 1 and Stage 2 rules*
2. *any Group 0 rules in Stage 3*

Proof: To verify that the complexity measure Δ_T is decreased by the application of any of the above group of rules, it is enough to note that the application of any of these rules reduces the size of some term $\overline{X} \dot{\sqsubseteq} \Gamma$ and leaves the size of others unchanged.

Hence, we have the proof.

Since Δ_T has the highest weight in the lexicographic ordering given by $K(C_s) = (\Delta_T, \Delta_{\dot{\sqsubseteq}}, \Delta_c)$, we have the following corollary.

Corollary B.0.5 *The measure $K(C_s) = (\Delta_T, \Delta_{\dot{\sqsubseteq}}, \Delta_c)$ is decreased by the application of:*

1. *any Stage 1 and Stage 2 rules*
2. *any Group 0 rules in Stage 3*

Lemma B.0.6 *The measure Δ_c is reduced by the application of any of the Group 1 rules in Stage 3.*

Proof: This claim is essentially identical to the termination claim for the language $\mathcal{L}_1$ except for rules **ExistsF** and **EqualsI**. It is easy to see that the complexity of each of the constraints that these rules act upon is reduced by these rules.

Hence we have the proof.

Now, we need to define the complexity measure $\Delta_{\dot{\sqsubseteq}}$ and show that the Group 2 rules in Stage 3 reduce this measure while at the same time we need to show that Group 1 rules (which reduce Δ_c) do not increase $\Delta_{\dot{\sqsubseteq}}$.

Definition B.0.7 *For a given constraint system C_s we define a binary relation **completes** as the least relation satisfying:*

- *x **completes** x*

- x ***completes*** X *if* $x \dot{\sqsubseteq} X \in C_s$
- x ***completes*** $\overline{C}$ *if if* $x \dot{\sqsubseteq} \overline{C} \in C_s$ *where* $\overline{C}$ *ranges over* $C, \neg C, \neg a, \neg c$
- x ***completes*** $\overline{Y} \sqcup \overline{Z}$ *if either* $x \equiv \overline{Y}, x \equiv \overline{Z}, x \dot{\sqsubseteq} \overline{Y} \in C_s$ *or* $x \dot{\sqsubseteq} \overline{Z} \in C_s$
- x ***completes*** $\exists f : \overline{Y}$ *if:*
  1. *either* $xfy \in C_s$ *or* $x \exists f\ y \in C_s$
     and
  2. *either* $y \dot{\sqsubseteq} \overline{Y} \in C_s$ *or* $y \equiv \overline{Y}$
- x ***completes*** $\forall f : \overline{Y}$ *if either of the following conditions hold:*
  1. $x \forall f\ y \in C_s$ *such that* $y \equiv \overline{Y}$ *or* $y \dot{\sqsubseteq} \overline{Y} \in C_s$
  2. $xfy \in C_s$ *such that* $y \equiv \overline{Y}$ *or* $y \dot{\sqsubseteq} \overline{Y} \in C_s$

We shall write x **waits_on** Γ if it is **not** the case that x **completes** Γ.

$\Delta_{\dot{\sqsubseteq}}$ is a multiset of pairs of complexity measures ordered lexicographically consisting of:

1. $(height(X), \Delta(x, \Gamma))$ for every occurrence of $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \Gamma \in C_s$
2. $(height(Y) + 1, \Delta(x, \Gamma))$ for every occurrence of $x \dot{\sqsubseteq} \Gamma \in C_s$ where Γ is of the form:
   (a) $\forall f : Y$
   (b) $z \sqcup Y$
   (c) $Y \sqcup z$

The individual measures $\Delta(x, \Gamma)$ are defined as follows for every possible shape of Γ:

- $\Delta(x, \forall f : \overline{Y}) = \{(0, 0, 0)\}$ if x **completes** $\forall f : \overline{Y}$
- otherwise $\Delta(x, \forall f : Y) =$
  $ancestors\_comp(x) \cup unsat\_daughters(x, \forall f : Y)$
- For every other constraint of the form Γ different from $\forall f : Y$ we have:
  - $\Delta(x, \Gamma) = \{(0, 0, 0)\}$ if x **completes** Γ
  - $\Delta(x, \Gamma) = \{(0, 0, 1)\}$ if x **waits_on** Γ

At this moment we shall refrain from defining the sets $ancestors\_comp(x)$ and $unsat\_daughters(x, \forall f : \overline{Y})$ associated with the rule **BForall**. Instead we shall show that assuming the complexity measure associated with rule **BForall** is not affected then all the other rules in Group 2 apart from rule **BForall** reduces the complexity measure.

Lemma B.0.8 *The application of all Group 2 rules in Stage 3 excluding rule* ***BForall*** *reduces the complexity measure* $\Delta_{\dot{\sqsubseteq}}$ *assuming that the application of these rules does not affect any individual measures of the form* $\Delta(x, \forall f : \overline{Y})$.

Proof: We shall prove the above lemma case by case for each individual rule involved.

1. Rule **BDis** : The application of this rule reduces the measure $(H_1, \Delta(x, \overline{X} \dot{\sqsubseteq} \overline{Y} \sqcup \overline{Z}))$ by ensuring that x **completes** $\overline{Y} \sqcup \overline{Z}$ while at the same time it introduces the new multiset of measures of the form $(H_2, \Delta(x, W \dot{\sqsubseteq} \Gamma))$ such that $height(H_1) > height(H_2)$. Secondly, the application of this rule does not change the complexity measure associated with other existing measures of the form $\Delta(x, \overline{X} \dot{\sqsubseteq} \Gamma)$. Hence, applying this rules reduces the measure $\Delta_{\dot{\sqsubseteq}}$.

2. Rule **BConst** : The application of this rule reduces the measure $\Delta(x, X \dot{\sqsubseteq} \overline{C})$ which in turn reduces the measure $\Delta_{\dot{\sqsubseteq}}$ in a manner analogous to the previous case.

3. Rule **BExists** : The application of this rule reduces the measure:
   $(H_1, \Delta(x, X \dot{\sqsubseteq} \exists f : \overline{Y}))$
   by ensuring that x **completes** $\exists f : \overline{Y}$ while at the same time it introduces the new multiset of measures of the form $(H_2, \Delta(y, \overline{Y} \dot{\sqsubseteq} \Gamma))$.

   Again, since the height of the introduced multisets is smaller than that of $(H_1, \Delta(x, X \dot{\sqsubseteq} \exists f : \overline{Y}))$ and since it is clear that it does not affect other measures in $\Delta_{\dot{\sqsubseteq}}$, the application of this rule decreases $\Delta_{\dot{\sqsubseteq}}$.

Hence we have the proof.

Since it is also clear that:

1. the above 3 rules do not increase Δ_T

2. Group 1 rules do not increase each of $\Delta(x, \overline{X} \dot{\sqsubseteq} \Gamma)$ as long as Γ is different from $\forall f : \overline{Y}$

we have the following lemma.

Lemma B.0.9 *The application of all Group 1 and Group 2 rules in Stage 3 excluding rule* ***BForall*** *reduces the complexity measure $K(C_s)$ assuming that:*

1. *the application of these rules do not affect any individual measures of the form $\Delta(x, \forall f : \overline{Y})$*
2. *Group 1 rules do not affect any individual measure of the form $\Delta(x, \forall f : \overline{Y})$*

Now, to prove the main theorem, we need to remove the above assumptions and show that

1. the measure $\Delta(x, \overline{X} \dot{\sqsubseteq} \forall f : \overline{Y})$ is not increased by any of the Group 1 and Group 2 rules
2. applying rule **BForall** decreases $\Delta(x, \overline{X} \dot{\sqsubseteq} \forall f : \overline{Y})$ while not increasing $\Delta_{\dot{\sqsubseteq}}$

We first need to define $\Delta(x, \overline{X} \dot{\sqsubseteq} \forall f : \overline{Y})$. For that purpose, we need to define the sets $ancestors\_comp(x)$ and $unsat\_daughters(x, \forall f : \overline{Y})$. We undertake this in the following definitions.

Definition B.0.10 [Path] *A primitive path relation between two variables x and y written $x \underset{P}{\rightarrow} y$ by overloading the terminology is defined as the least relation satisfying the following conditions:*

1. $x \underset{\epsilon}{\rightarrow} y$ *if any of* $x = y \in C_s$, $y = x \in C_s$, $x \dot{\sqsubseteq} y \in C_s$ *or* $y \dot{\sqsubseteq} x \in C_s$
2. $x \underset{f}{\rightarrow} y$ *if any of* $xfy \in C_s$, $x\ \exists f\ y \in C_s$ *or* $x\ \forall f\ y \in C_s$
3. $x \underset{\epsilon}{\rightarrow} y$ *if any of the following conditions hold:*
   - $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} y \in C_s$
   - $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} y \sqcup \overline{Z} \in C_s$ *and* x ***waits_on*** $X \dot{\sqsubseteq} y \sqcup \overline{Z}$
   - $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \overline{Z} \sqcup y \in C_s$ *and* x ***waits_on*** $X \dot{\sqsubseteq} \overline{Z} \sqcup y$
   - $x \dot{\sqsubseteq} y \sqcup \overline{Z} \in C_s$ *and* x ***waits_on*** $y \sqcup \overline{Z}$
   - $x \dot{\sqsubseteq} \overline{Z} \sqcup y \in C_s$ *and* x ***waits_on*** $\overline{Z} \sqcup y$
4. $x \underset{P'}{\rightarrow} y$ *if* $P \not\equiv \sqcup$, P' *is obtained from* P *by replacing every occurrence of* $\sqcup$ *by* ϵ *and any of the following conditions hold:*

   In the following the reader is referred to definition 2.7.3 for a definition of the path relation $X \underset{P}{\rightarrow} Y$ between big variables.

   - $x \dot{\sqsubseteq} X, X' \dot{\sqsubseteq} y \in C_s$ *and* $X \underset{P}{\rightarrow} X'$

- $x \dot{\sqsubseteq} X, X' \dot{\sqsubseteq} y \sqcup \overline{Z} \in C_s$ *and* $X \underset{P}{\rightarrow} X'$
- $x \dot{\sqsubseteq} X, X' \dot{\sqsubseteq} \overline{Z} \sqcup y \in C_s$ *and* $X \underset{P}{\rightarrow} X'$
- $x \dot{\sqsubseteq} y \sqcup \overline{Z} \in C_s$ *and* $X \underset{P}{\rightarrow} X'$
- $x \dot{\sqsubseteq} \overline{Z} \sqcup y \in C_s$ *and* $X \underset{P}{\rightarrow} X'$

5. $x \underset{P'.f}{\rightarrow} y$ *if* P' *is obtained from* P *by replacing every occurrence of* $\sqcup$ *by* ϵ *and any of the following conditions hold:*
   - $x \dot{\sqsubseteq} X, X' \dot{\sqsubseteq} F : y \in C_s$ *and* $X \underset{P}{\rightarrow} X'$ *where* F *ranges over* $f, \exists f, \forall f$
6. *the empty path relation* $\underset{\epsilon}{\rightarrow}$ *is transitive and reflexive*

Definition B.0.11 [Acyclic Path] *We say that a path* P *connects a variable* x_0 *to another variable* x_n ***acyclically*** *written* $x_0 \underset{P}{\Rightarrow} x_n$ *if* P *is a non-empty path and the there exists a non-repeating sequence of variables* $x_0, \ldots, x_n$ *such that* $x_0 \underset{p_1}{\rightarrow} x_1, x_1 \underset{p_2}{\rightarrow} x_2, \ldots, x_{n-1} \underset{p_n}{\rightarrow} x_n$ *where* $P = p_1 \ldots . p_n$.

Since any finite constraint system contains a finite number of variables, we have the following proposition.

Proposition B.0.12 *For a given constraint system every acyclic path between any two variables is of finite length.*

For any given pair of variables (x, y) we define *all_paths*(x, y) to be the set of all *acyclic paths* that connect x to y. Since there are a finite number of variables, big variables and paths that relate big variables we have the following proposition.

Proposition B.0.13 *For a finite constraint system* C_s *the set all_paths*(x, y) *for any two variables* x, y *in* C_s *is finite.*

Let $succ(x, f)$ denote the set of f-successors of x given by:

$$succ(x, f) = \{y \mid xFy \in C_s \text{ where } F \text{ ranges over } f, \exists f, \forall f\}$$

Let $|succ(x, f)|$ denote the cardinality of $succ(x, f)$.

Let *ancestor_daughters*(x) denote the *multiset* of all pairs $(length(P), |succ(x', f)|)$ such that f is any relational symbol and x' connects via an acyclic path P to x:

$$ancestor\_daughters(x) = \{(length(P), |succ(x', f)|) \mid succ(x', f) \neq \emptyset \ \wedge x' \underset{P}{\Rightarrow} x \text{ in } C_s\}$$

Each pair $(length(P), |succ(x', f)|)$ in the set *ancestor_daughters*(x) is a measure of the number of f-successors at a acyclic distance P from x.

Let C_s and C'_s be any two constraint systems and let $ancestor\_daughters(x)$ and $ancestor\_daughters'(x)$ be the corresponding ancestor daughter multisets for any given variable x. Consider the multiset ordering $\succ$ on multisets $ancestor\_daughters(x)$ and $ancestor\_daughters'(x)$ induced by the lexicographic ordering on the member tuples. The following lemma will play a fundamental role in proving the termination claim.

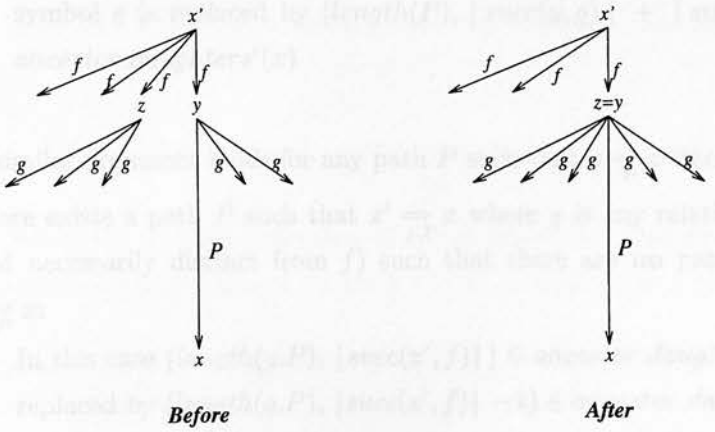


Figure B.22: Effect of applying rule 5 to a constraint system

Lemma B.0.14 *Let C_s be any constraint system and let x be any variable in C_s. Let the application of any Group 1 rule transform C_s into C'_s. Then*

Either $ancestor\_daughters(x) = ancestor\_daughters'(x)$

or $ancestor\_daughters(x) \succ ancestor\_daughters'(x)$

Proof: We shall prove the above lemma by case by case analysis of the effect of each rule on the multiset ordering $\succ$.

1. Rule **Equals** : This rule rewrites variable y to x' such that both $x' \underset{\epsilon}{\rightarrow} y$ and $y \underset{\epsilon}{\rightarrow} x'$. The path relation $\underset{P}{\Rightarrow}$ remains unaffected. Similarly, the the number of f-successors of every variable including x' and y remain unaffected. Hence for any variable x in C_s:
   $ancestor\_daughters(x) = ancestor\_daughters'(x)$

2. Rule **Feat** : This rule rewrites $x'fy, x'Fz$ to $x'fy, y = z$.
   We need to examine the following cases:

   (a) There exists a path P such that $y \underset{P}{\Rightarrow} x$; then we know that $x' \underset{f.P}{\Rightarrow} x$ (see figure B.22):

In this case $(length(f.P), |succ(x', f)|) \in ancestor\_daughters(x)$ gets replaced by
$(length(f.P), |succ(x', f)| - 1) \in ancestor\_daughters'(x)$, while at the same time
$(length(P), |succ(y, g)|) \in ancestor\_daughters(x)$ for any relational symbol g is replaced by $(length(P), |succ(y, g)| + |succ(z, g)|) \in ancestor\_daughters'(x)$

(b) A similar argument holds for any path P such that $z \underset{P}{\Rightarrow} x$ since $x' \underset{f.P}{\Rightarrow} x$.

(c) There exists a path P such that $x' \underset{g.P}{\Rightarrow} x$ where g is any relational symbol (not necessarily distinct from f) such that there are **no** paths $y \underset{P}{\Rightarrow} x$ or $z \underset{P}{\Rightarrow} x$:

In this case $(length(g.P), |succ(x', f)|) \in ancestor\_daughters(x)$ gets replaced by $(length(g.P), |succ(x', f)| - 1) \in ancestor\_daughters'(x)$.

(d) Every other pair $(length(P), |succ(x, f)|)$ for arbitrary P and f distinct from the above cases is unaffected by the application of this rule.

From the above cases we either get:
$ancestor\_daughters(x) \succ ancestor\_daughters'(x)$ or we get:
$ancestor\_daughters(x) = ancestor\_daughters'(x)$.

3. Rule **ExForall** : This case is analogous to the previous case.

4. Rules **ExistsF**,**EqualsE** and **Neg** : In each of these cases it is obvious that $ancestor\_daughters(x) = ancestor\_daughters'(x)$

Hence, we have proved the lemma.

The above lemma demonstrates the fact that the application of Group 1 rules decreases the number of f-successors for ancestors at larger "distances" while possibly increasing the number of f-successors for ancestors at shorter distances. This will be a fundamental property we shall utilise in proving termination.

For the purposes of the termination proof we further need to take into account the new constraints that could potentially be generated by the application of rule **BExists**. For this purpose, in the following definitions we introduce the measure $\Pi_\exists(x, f)$ denoting the number of existential constraints of the form $\exists f : \overline{Y}$ yet to be **completed** by ancestors of x.

Let $\Pi_\exists(x, f)$ be the set of "**waits_on**" constraints of the form $\exists f : \overline{Y}$ given by:

$$\Pi_\exists(x, f) = \{X \dot{\sqsubseteq} \exists f : \overline{Y} \mid x \dot{\sqsubseteq} X \in C_s \wedge X \dot{\sqsubseteq} \exists f : \overline{Y} \in C_s \wedge x \textbf{ waits\_on } \exists f : \overline{Y}\}$$

Let $|\Pi_\exists(x, f)|$ denote the cardinality of the set $\Pi_\exists(x, f)$.
Similarly, let $\Pi_\forall(x, \forall f : \overline{Y})$ be the set of daughters of x which have not satisfied $\forall f : \overline{Y}$ given by:

$$\Pi_\forall(x, \forall f : \overline{Y}) = \{y' \mid x \underset{\epsilon}{\longrightarrow} x' \text{ in } C_s \wedge x' F y' \in C_s \wedge F \in \{f, \exists f\} \wedge x' \textbf{ waits\_on } \overline{Y}\}$$

Let $\Pi_\forall(x, f)$ denote the multiset consisting of all the unsatisfied daughters of x which have not satisfied every constraint of the form $\forall f : \overline{Y}$ "attached" to x given by:

$$\Pi_\forall(x, f) = \bigcup_{x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \forall f : \overline{Y} \in C_s} \Pi_\forall(x, \forall f : Y) \cup \bigcup_{x \dot{\sqsubseteq} \forall f : Y \in C_s} \Pi_\forall(x, \forall f : Y)$$

Let $ancestor\_comp(x)$ denote the multiset consisting of triples $(length(P), |\Pi_\exists(x', f)|, |\Pi_\forall(x', f)| + |succ(x', f)|)$ for every x' such that $x' \underset{P}{\Longrightarrow} x$ given by:

$$ancestor\_comp(x) = \{(length(P), |\Pi_\exists(x')|, |\Pi_\forall(x', f)| + |succ(x', f)|) \mid x' \underset{P}{\Longrightarrow} x \text{ in } C_s \wedge |\Pi_\exists(x', f)| + |\Pi_\forall(x', f)| + |succ(x', f)| > 0\}$$

Lemma B.0.15 *Let C_s be any constraint system and let x be any variable in C_s. Let the application of any Group 1 rule transform C_s into C_s'. Then*

Either $ancestor\_comp(x) = ancestor\_comp'(x)$

or $ancestor\_comp(x) \succ ancestor\_comp'(x)$

Proof: The above lemma can be proved in an analogous fashion to the lemma B.0.14.

Let $unsat\_daughters(x, \forall f : \overline{Y})$ denote the triple $(0, \Pi_\exists(x, f), \Pi_\forall(x, \forall f : \overline{Y}))$. In other words, the set $unsat\_daughters(x, \forall f : \overline{Y})$ provides a measure of the number of f-successors of x that are *waiting* on $\forall f : \overline{Y}$.

Our definitions imply that every member (N, A, B) in $ancestor\_comp(x)$ is greater than every member $(0, A', B') \in unsat\_daughters(x, \forall f : \overline{Y})$ purely because $N > 0$ always holds. This is so because our definition of *acyclic path* only considers paths of length greater than 0.

Recollect that the complexity measure $\Delta(x, \forall f : Y)$ is given by:

$$ancestors\_comp(x) \cup unsat\_daughters(x, \forall f : Y)$$

where $ancestors\_comp(x)$ and $unsat\_daughters(x, \forall f : Y)$ are as defined above.

Lemma B.0.16 *The application of any Group 1 rule does not increase* $\Delta_{\dot{\sqsubseteq}}$

Proof: From lemma B.0.15 we know that the size of $ancestor\_comp(x)$ for every variable x in C_s does not increase by the application of Group 1 rules.

To prove the above claim, we need to show that when Group 1 rules are applied *either* of the following hold:

1. the size of $unsat\_daughters(x, \forall f : \overline{Y})$ does not increase for every variable x in C_s
2. if the size of $unsat\_daughters(x, \forall f : \overline{Y})$ increases then this is accompanied by a decrease in the size of $ancestor\_comp(x)$ with the effect that $\Delta_{\dot{\sqsubseteq}}$ is decreased.

For rules **Equals**, **ExistsF**, **EqualsE** and **Neg** it is quite clear that the size of both $\Pi_{\exists}(x, f)$ and
$unsat\_daughters(x, \forall f : \overline{Y})$ are unaffected by these rules. This means that the measure $\Delta(x, \forall f : Y)$ is unaffected. Since it also clear that each of these rules do not affect any other measure in $\Delta_{\dot{\sqsubseteq}}$, it follows that applying any of these rules does not increase $\Delta_{\dot{\sqsubseteq}}$.

It remains to verify this with respect to rules **Feat** and **ExForall**.

Assume that the application of either of the rules **Feat** or **ExForall** rewrites $xFy,\ xF'z$ by $xfy,\ y = z$ where F, F' ranges over $f, \exists f, \forall f$.

For both the rules **Feat** and **ExForall** it is clear that the size of $unsat\_daughters$ can potentially increase only for the variables y and z. In other words, only potentially the sizes of $unsat\_daughters(y, \forall f : \overline{W_1})$ and $unsat\_daughters(z, \forall f : \overline{W_2})$ can increase for some feature f and variables $\overline{W_1}$ and $\overline{W_2}$. Hence, to verify the lemma it is enough to show that whenever the size of either $unsat\_daughters(y, \forall f : \overline{W_1})$ or $unsat\_daughters(z, \forall f : \overline{W_2})$ increases, this is accompanied by a decrease in the corresponding sizes of $ancestor\_comp(y)$ and $ancestor\_comp(z)$ respectively.

For variables, y and z we know that the application of either of the rules **Feat** or **ExForall** decreases both $ancestor\_comp(y)$ and $ancestor\_comp(z)$ by decreasing the number of f-successors of x which is an ancestor of both y and z at distance 1.

Hence, we have the lemma.

Since the application of Group 1 rules does not affect the measure Δ_T and from the above lemma it does not affect the measure $\Delta_{\dot{\sqsubseteq}}$ we have the following corollary.

Corollary B.0.17 *The application of any of the Group 1 rules decreases the complexity measure $K(C_s)$.*

Lemma B.0.18 *Taking into account our rule ordering the application of the Group 2 rule **BExists** does not increase existing measures of the form $\Delta(w, \forall f : \overline{W_1})$ in such a way as to increase $\Delta_{\dot{\sqsubseteq}}$.*

Proof: In order to verify this claim we need to take into account the interaction of the new variable generated by the application of rule **BExists** with existing variables.

Assume that the application of rule **BExists** rewrites $x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \exists f : \overline{Y}$ by $x \exists f y, y \dot{\sqsubseteq} \overline{Y}, x \dot{\sqsubseteq} X, X \dot{\sqsubseteq} \exists f : \overline{Y}$.

We need to consider 2 cases:

Case 1: Assume that for every variable z such that $y \underset{\epsilon}{\longrightarrow} z$ we have $z \equiv y$.

Let w be a variable in C_s such that $y \underset{P}{\Longrightarrow} w$. Let $\Delta(w, \forall f : \overline{W_1})$ be a complexity measure in $\Delta_{\dot{\sqsubseteq}}$.

We claim that the size of $ancestor\_comp(w)$ decreases.

We know that the size of $\Pi_\exists(x, f)$ decreases by 1 while the size of $succ(x, f)$ increases by 1. In effect, the size of the multiset complexity $(length(P) + 1, |\Pi_\exists(x)| + |succ(x, f)|)$ decreases by 1. At the same time, a number of new multiset complexities of the form $(length(P), |\Pi_\exists(y)| + |succ(y, f)|)$ gets added. Since $length(P) + 1 > length(P)$, we know that the measure $ancestor\_comp(w)$ decreases.

This means that the measure $\Delta(w, \forall f : \overline{W_1})$ decreases.

Case 2: Assume that there exists a variable z such that $y \underset{\epsilon}{\longrightarrow} z$ such that $y \not\equiv z$. Crucially taking into account our rule ordering and the definition of $\underset{\epsilon}{\longrightarrow}$, the only way this could arise is by one of the following:

(a) $\overline{Y} \equiv z$

(b) $\overline{Y} \equiv Y$ and $Y \dot{\sqsubseteq} z \in C_s$

(c) $\overline{Y} \equiv Y$, $Y \dot{\sqsubseteq} \overline{Z_1} \sqcup \overline{Z_2} \in C_s$ and

- either $z \equiv \overline{Z_1}$

- or $z \equiv \overline{Z_2}$

(d) $\overline{Y} \equiv y'$, $y' \dot{\sqsubseteq} \overline{Z_1} \sqcup \overline{Z_2} \in C_s$ and

- either $z \equiv \overline{Z_1}$
- or $z \equiv \overline{Z_2}$

In each case the number of $unsat\_daughters(z, \forall f' : \overline{Z'})$ remains the same since y is a new variable and hence there are no constraints of the form yfz' or $y\ \exists f\ z'$ in C_s.

For every variable w such that $y \underset{P}{\Rightarrow} w$ we know that $x \underset{f.P}{\Rightarrow} w$ and hence any complexity measure of the form $(length(P) + 1, \mid \Pi_\exists(x, f) \mid + \mid succ(x, f) \mid)$ decreases while potentially adding smaller measures of the form $(length(P), \mid \Pi_\exists(y, f') \mid, \mid \Pi_\forall(y, f') \mid + \mid succ(y, f') \mid)$.

For every variable w such that $z \underset{P}{\Rightarrow} w$, we know that $x \underset{f.P}{\Rightarrow} w$ and a similar reasoning as for $y \underset{P}{\Rightarrow} w$ applies.

In effect, this means that the size of $ancestor\_comp(w)$ decreases.

This means that the measure $\Delta(w, \forall f' : \overline{Z'})$ decreases.

Hence, we have the lemma.

Lemma B.0.19 *The application of the Group 2 rule* ***BExists*** *decreases* $\Delta_{\dot{\sqsubseteq}}$.

Proof: Since it is clear that the application of rule **BExists** does not increase existing complexity measures so as to increase $\Delta_{\dot{\sqsubseteq}}$ while at the same time it decreases one of the measures $\Delta(x, \exists f : \overline{Y})$ which means that it decreases $\Delta_{\dot{\sqsubseteq}}$.

Lemma B.0.20 *The application of the Group 2 rule* ***BForall*** *does not increase existing measures of the form* $\Delta(w, \forall f : \overline{W'})$ *in such a way as to increase* $\Delta_{\dot{\sqsubseteq}}$.

Proof: Let rule **BForall** apply to the constraint xFy to add $y \dot{\sqsubseteq} \overline{Y}$ where F ranges over $f, \exists f$.

Taking into account our rule ordering, it is clear that it decreases the size of $unsat\_daughters(x, \forall f : \overline{Y})$. This means it decreases the size of $(H, \Delta(x, \forall f : \overline{Y}))$ while adding a finite number of tuples of the form $(H - 1, \Delta(y, \Gamma))$ can get added.

We need to show that this does not affect existing measures of the form $\Delta(w, \forall f : \overline{W'})$ in such a way as to increase $\Delta_{\dot{\sqsubseteq}}$.

Let w be a variable such that $y \underset{P}{\Rightarrow} w$, then we know that $x \underset{f.P}{\Rightarrow} w$. The application of rule **BForall** reduces the measure $\Pi_\forall(x, f)$ which means it reduces the measure

$(length(P) + 1, \mid \Pi_{\exists}(x, f) \mid, \mid \Pi_{\exists}(x, f) \mid + \mid succ(x, f) \mid)$. Note that the measures $\mid \Pi_{\exists}(x, f) \mid$ and $\mid succ(x, f) \mid$ are not affected. At the same time, applying this rule could potentially add new complexity measures of the form $(length(P), \Gamma)$. Since $length(P) + 1 > length(P)$, we know that this addition cannot increase the measure $\Delta_{\dot{\sqsubseteq}}$.

Hence, we have the lemma.

Lemma B.0.21 *The application of the Group 2 rule* ***BForall*** *decreases* $\Delta_{\dot{\sqsubseteq}}$.

Proof: Since it is clear that the application of rule **BForall** does not increase existing complexity measures so as to increase $\Delta_{\dot{\sqsubseteq}}$ while at the same time it decreases one of the measures $\Delta(x, \forall f : \overline{Y})$ which means that it decreases $\Delta_{\dot{\sqsubseteq}}$.

Lemma B.0.22 *The application of the any Group 2 rules decreases* $\Delta_{\dot{\sqsubseteq}}$.

Since both rule **BForall** and rule **BExists** decrease $\Delta_{\dot{\sqsubseteq}}$ and since it is clear that the other rules reduce $\Delta_{\dot{\sqsubseteq}}$, we have the lemma.

Lemma B.0.23 *The application of the any Group 2 rules decreases* $K(C_s)$.

Since none of the Group 1 rules affect $\Delta_{\dot{\sqsubseteq}}$, while Group 0 rules decrease Δ_T and Group 2 rules decrease $\Delta_{\dot{\sqsubseteq}}$ the above claim follows.

Since the application of any rule in each group decreases $K(C_s)$ we have the following claim.

Corollary B.0.24 *The application of any rule decreases* $K(C_s)$.

This means that our normalisation rules terminate.

Hence we have the theorem.